HF44b0使用手册

版本 4.0

北京恒丰锐科科技有限公司

目录

附加：A

第一章 概述

HF44b0开发板是基于三星公司S3C44b0x 高性能ARM 处理器的嵌入开发平台，旨在为用户提供完整的嵌入式入门解决方案。

硬件规格：

1. Samsung S3C44B0X 66MHz (ARM7 内核)
2. 8M SDRAM
3. 2M FLASH
4. 10M 以太网接口
5. IDE 接口
6. USB1.1 接口
7. 音频输出
8. MIC 输入接口
9. 时实时钟，备有可充电电池，方便您的使用。
10. IIC 总线接口
11. 实时系统时钟
12. LCD 接口：支持 640*480 以下单色或 320*240 以下 STN/DSTN 256 色
13. 2 个 RS232 串行口
14. 包括 protel 格式的原理图，有利您硬件的学习。
15. 20 针 JTAG 调试端口
16. 1*4 键盘
17. 可扩展的总线接口,CPU 的 160 个引脚全部用 2.54 标准插座引出，您可以任意扩展你的设想，我们也逐步推出可选组件（保持向下的兼容性，保护你的投资）供你选择。

第一节 CPU 介绍

1.1 简介

Samsung 公司推出的 16/32 位 RISC 处理器 S3C44B0X 为手持设备和一般类型应用提供了高性价比和高性能的微控制器解决方案。为了降低成本，S3C44B0X 提供了丰富的内置部件：8KB Cache，可选的内部 SRAM，LCD 控制器，带自动握手的 2 通道 UART，4 通道 DMA，系统管理器（片选逻辑，FP/EDO/SDRAM 控制器），带 PWM 功能 5 通道定时器，I/O 端口，RTC，8 通道 10 位 ADC，IIC 总线接口，IIS 总线接口，同步 SIO 接口和 PLL 倍频器。

S3C44B0X采用ARM7TDMI 内核，0.25um工艺的CMOS标准宏单元和存储编译器。它低功耗，精简，出色和全静态的设计特别适用于对成本和功耗敏感的应用。同样S3C44B0X还采用了一种新的总线结构，即SAMBA II（SAMSUNG ARM CPU 嵌入式微处理器总线结构）。

S3C44B0X 的显著特性是它的 CPU 核，是由 ARM 公司设计的 16/32 位 ARM7TDMI RISC 处理器（66MHZ）。ARM7TDMI 体系结构的特点是它集成了 Thumb 代码压缩器，一个片上 ICE 断点调试支持，和一个 32 位的硬件乘法器。

S3C44B0X 通过提供全面的、通用的片上外设，大大减少了系统电路中除处理器以外的元器件配置，

从而最小化系统的成本。片上集成的主要功能如下：

★ 2.5V ARM7TDMI 内核，带有 8K Cache（SAMBA II 总线体系结构，主频高至 66MHz）；
★ 外部存储器控制器（FP/EDO/SDRAM 控制，片选逻辑）；
★ LCD 控制器(最大支持 256 色 DSTN)，并带有 1 通道 LCD 专用 DMA；
★ 2 通道通用 DMA、2 通道外设 DMA 并具有外部请求引脚；
★ 2 通道 UART 带有握手协议(支持 IrDA1.0, 具有 16-byte FIFO)/1 通道 SIO；
★ 1 通道多主 IIC-BUS 控制器；
★ 1 通道 IIS-BUS 控制器；
★ 5 个 PWM 定时器和 1 个内部定时器；
★ 看门狗定时器；
★ 71 个通用 I/O 口/8 通道外部中断源；
★ 功耗控制：具有正常、低速、空闲和停止模式；
★ 8 通道 10 位 ADC；
★ 具有日历功能的 RTC；
★ 带 PLL 的片上时钟发生器。

1.2 特性

体系结构

* 集成了手持设备和通用嵌入式系统应用的解决方案；
* 16/32 位 RISC 体系结构和 ARM7TDMI 处理器内核强大的指令体系；
* Thumb 代码压缩机，最大化代码密度同时保持了 32 位指令的性能；
* 基于 JTAG 的片上集成 ICE 调试支持解决方案；
* 32×8 位硬件乘法器；
* 实现低功耗 SAMBA II（三星 ARM 处理器嵌入式微控制器总线体系结构）的新型总线结构。

系统管理器

* 支持大/小端模式；
* 地址空间：每 bank 为 32M 字节（共 256M 字节）；
* 支持每 bank 可编程的 8/16/32 位数据总线宽度；
* 7 个 bank 具有固定的 bank 起始地址和可编程的 bank 大小；
* 1 个 bank 具有可编程的 bank 起始地址和 bank 大小；
* 8 个存储器 bank：
  −6 个 ROM，SRAM 存储器 bank；
  −2 个 ROM/SRAM/DRAM（快速页面，EDO 和同步 DRAM）；
* 所有的存储器 bank 具有可编程的访问周期；
* 支持外部等待信号延长总线周期；
* 支持掉电时 DRAM/SDRAM 的自刷新模式；
* 支持均匀/非均匀的 DRAM 地址。

Cache 存储器和内部 SRAM

* 一体化的 8k 字节 cache；
* 未用的 cache 空间用来作为 0/4/8 k 字节的 SRAM 存储空间；
* 支持 LRU（近期最少使用）替换算法；
* 采用保持主存储器与 cache 内容一致性的“写穿式”策略；
* 写缓冲具有 4 级深度；
* 当缓冲区出错时，采用“请求数据优先填充”技术。

时钟和电源管理

* 低功耗；
* 片上 PLL 使 MCU 工作时钟最大达到 75MHz；
* 可以通过软件设置各功能模块的输入时钟；
* 电源模式：正常，慢速，空闲和停止模式；
  正常模式：正常工作模式；
  低速模式：不带 PLL 的低时钟频率模式；
  空闲模式：只停止 CPU 的时钟；
  停止模式：停止所有的时钟；
* 通过 EINT[7:0]或 RTC 报警中断从停止模式唤醒。

中断控制器

* 30 个中断源（1 个看门狗定时器，6 个定时器，6 个 UART，8 个外部中断，4 个 DMA，2 个 RTC，1 个 ADC，1 个 IIC，1 个 SIO）
* 采用向量 IRQ 中断模式以缩短中断响应时间；
* 可选的电平/边沿模式触发外部中断；
* 电平/边沿模式具有可编程的极性；
* 支持紧急中断请求的 FIQ（快速中断请求）。

带 PWM 的定时器（脉宽可调制）

* 5 个 16 位带 PWM 功能的定时器，1 个 16 位内部定时器（可进行基于 DMA 或基于中断的操作）；
* 可编程的占空比，频率，和极性；
* 死区产生器；
* 支持外部时钟源。

实时时钟 RTC

* 全时钟特性：毫秒、秒、分钟、小时、日、星期、月、年；
* 32.768KHz 时钟；
* 用于唤醒 CPU 的报警中断；
* 时钟节拍中断。

通用 I/O 口

* 8 个外部中断端口；
* 71 个（多功能）复用输入/输出口。

UART(异步串行通讯)

* 2 通道 UART，可进行基于 DMA 或基于中断的操作；
* 支持 5 位，6 位，7 位，8 位串行数据发送/接收；
* 支持在发送/接收期间的 H/W 握手功能；
* 可编程的波特率；
* 支持 IrDA1.0（115.2Kbps）；
* 支持用于测试的回馈模式；
* 每个通道具有 2 个内部 32 字节的 FIFO 分别用于接收和发送。

DMA（直接存储器操作）控制器

* 2 通道通用的无需 CPU 干涉的 DMA 控制器；
* 2 通道桥式 DMA（外设 DMA）控制器；
* 支持 I/O 到存储器，存储器到 I/O，I/O 到 I/O 的 6 种 DMA 请求：软件，4 个内部功能模块（UART，SIO，定时器，IIS），和外部引脚；
* DMA 之间的优先级次序可编程；
* 采用猝发式的传输模式以提高 FPDRAM, EDODRAM 和 SDRAM 的数据传输速率；
* 支持在外部设备到存储器和存储器到外部设备之间采用 fly－by 模式。

A/D 转换器

＊ 8 通道的 ADC；

＊ 最大 100k SPS/10-bit。

LCD 控制器

＊ 支持彩色/黑白/灰度 LCD；

＊ 支持单路扫描和双路扫描；

＊ 支持虚拟显示功能；

＊ 系统存储器用来作为显示缓存；

＊ 用专门的 DMA 来从系统存储器中获得图象数据；

＊ 可编程屏幕大小；

＊ 灰度等级：16 级；

＊ 彩色模式：256 色。

看门狗定时器

＊ 16 位看门狗定时器；

＊ 在定时器溢出时发出中断请求或系统复位。

IIC 总线接口

＊ 1 通道多主 IIC 总线，可进行基于中断的操作；

＊ 8 位，串行，双向数据传输，标准模式速度达到 100Kbit/s，快速模式达到 400Kbit/s。

IIS 总线接口

＊ 1 通道音频 IIS 总线接口，可进行基于 DMA 的操作；

＊ 每通道 8/16 位串行数据传输；

＊ 支持 MSB-justified 数据格式。

SIO（同步串行 I/O）

＊ 1 通道 SIO，可进行基于 DMA 或基于中断的操作；

＊ 可编程的波特率；

＊ 支持 8 位 SIO 的串行数据发送和接收操作。

工作电压范围

＊ 内核：2.5V； I/O：3.0V 到 3.6V。

工作频率

＊ 最大为 66MHz。

封装

＊ 160LQFP/160FBGA

第二节接口介绍

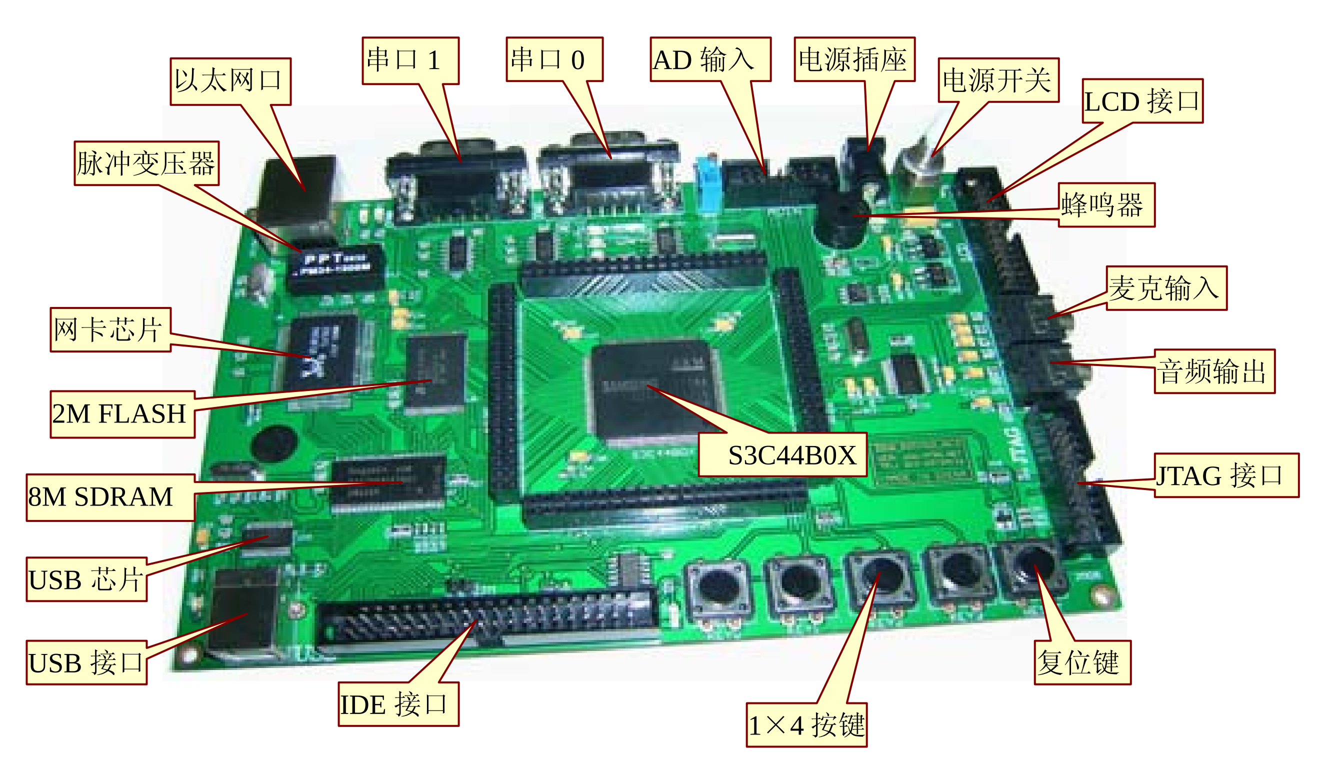


图 1-1 hf44b0 开发板结构图

Hf44b0 开发板提供了丰富的接口资源，共大家学习研究。我们分别对这些接口做一下介绍，

IDE 接口：

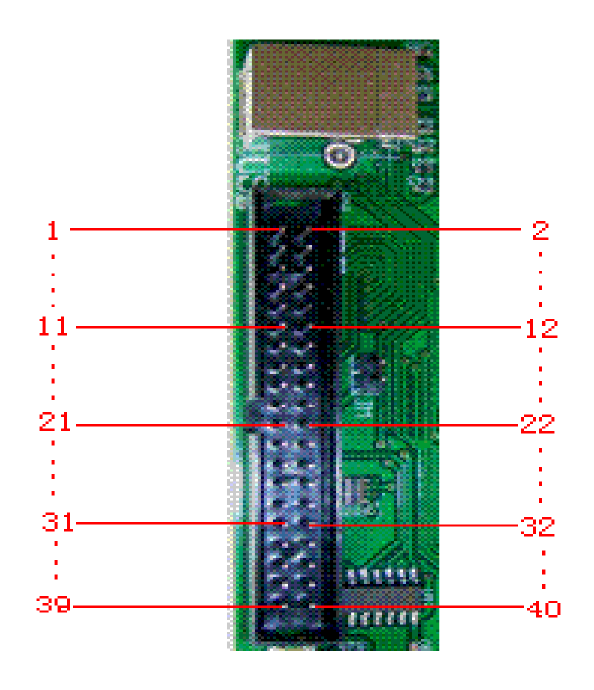


图 1-2 IDE 接口

引脚	名称	引脚	名称	引脚	名称
1	ATA_RST	2	GND	3	DATA7
4	DATA8	5	DATA6	6	DATA9
7	DATA5	8	DATA10	9	DATA4
10	DATA11	11	DATA3	12	DATA12
13	DATA2	14	DATA13	15	DATA1
16	DATA14	17	DATA 0	18	DATA 15
19	GND	20	VCC	21	ATA_DMAREQ
22	GND	23	ATA_DIOW	24	GND
25	ATA_DIOR	26	GND	27	ATA_IORDY
28	GND	29	ATA_DMACK	30	GND
34	NC	32	GND	33	ADDR2
35	ADDR1	36	ADDR3	37	ADDR4
38	ADDR5	39	ATA_DASP	40	GND

JTAG 接口：

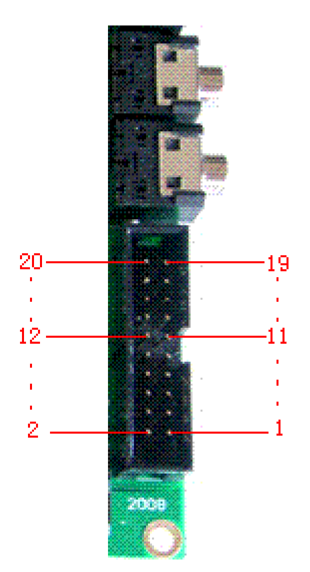


图 1-3 JTAG 接口

引脚	名称	引脚	名称	引脚	名称
1	VCC	2	VCC	3	nTRST
4	GND	5	TDI	6	GND
7	TMS	8	GND	9	TCK
10	GND	11	GND	12	GND
13	TDO	14	GND	15	nRESET
16	GND	17	NC	18	GND
19	NC	20	GND		

LCD 接口：

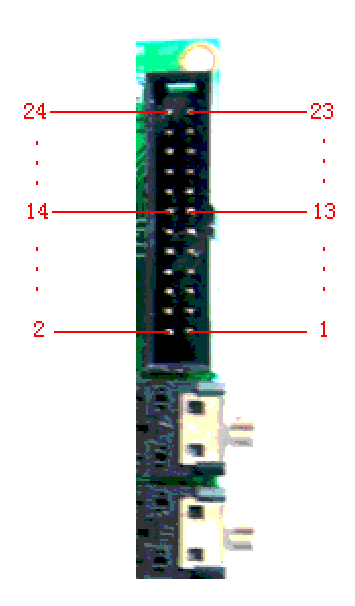


图 1-4 LCD 接口

引脚	名称	引脚	名称	引脚	名称
1	VD0	2	VD1	3	VD2
4	VD3	5	VD4	6	VD5
7	VD6	8	VD7	9	VFRAME
10	GND	11	VCLK	12	VLINE
13	VM	14	nRESET	15	EL_ON
16	DISP-ON	17	GPE3	18	GPE4
19	GPE6	20	GPE7	21	KEYIN3
22	VCC	23	VCC5.0	24	VCC5.0

AD 输入：

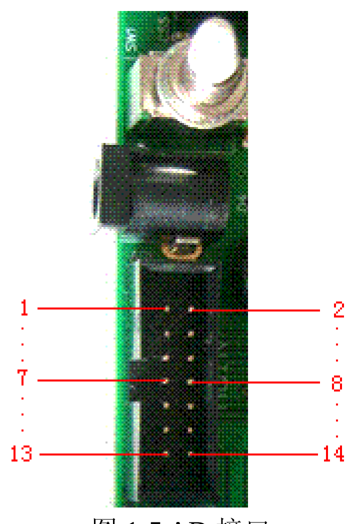


图 1-5 AD 接口

引脚	名称	引脚	名称	引脚	名称
1	AIN	2	AIN0	3	AIN1
4	AIN0	5	AIN3	6	AIN2
7	AIN5	8	AIN4	9	AIN7
10	AIN6	11	NC	12	NC
13	GND	14	VCC		

第三节地址分配

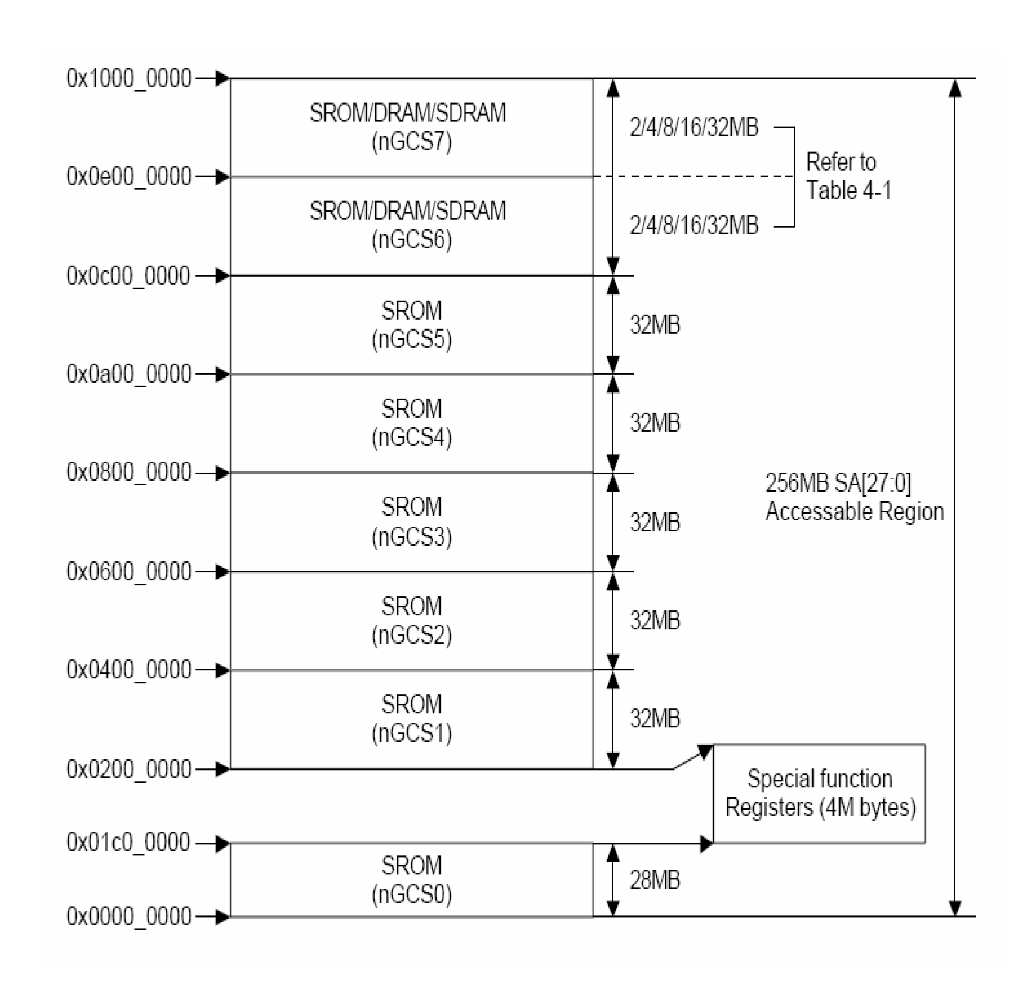


图 1-6 内存分配图

44b0 外围芯片的地址由使用的 44b0 的片选决定，每个片选分配的地址如图 1-6 所示。

比如 nor flash 芯片 AD19lv160 使用的是片选 0，所以他的开始地址是 0，芯片是 2M，所以整个 nor flash 的地址为：0x0----0x1fffff 之间。

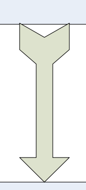
比如使用的 SDRAM 是 hy57V641620，接到了片选 6，所以开始地址为 0x0c000000,sdarm 的大小为 8M，所以 SDRAM 的地址范围为：0x0c000000---0x0c7fffff

网卡芯片 RTL8019 接到了片选 3，所以起始地址是 0x06000000.详细的网卡寄存器地址可以通过了解网卡的芯片手册来获得。

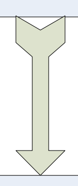
第四节如何来学习 arm

Arm 开发的整体流程起始并不复杂，很多初级入门的客户往往不清楚怎么来入门。我们在这里简要的说明一下。

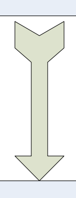
基础知识：
1.基本的外语知识，可以看懂英文手册，44b0现在有部分书籍可以参考，但是绝大部分手册还是英文的，所以我还是看懂的好啊
2.良好的C语言基础，绝大部分底层代码都是C语言所写，所以你必须精通C语言
3.学会ARM的汇编语言，虽然汇编的程序不是很多，但是也是必不可少的部分，所以你最少要看的懂
4.原理图的读图能力，虽然你的目的不是硬件的学习，但是你最少要看的懂原理图。
5.良好的计算机使用能力，因该熟习必要常见的网络感念和常用的软件操作，除了基本的windows以外，常用的还有linux，这个在一般的嵌入式开发中也必不可少。
准备好这些基础知识就可以开始学习arm的开发了



第一阶段的基本操作的学习：
在这个阶段主要学习如何使用这个开发板，熟悉开发板的硬件接口，需要掌握的内容有：
1。了解硬件的基本机构和看懂原理图，浏览44b0的手册，弄清开发板的地址关系
2。学习u-boot的基本操作，比如下载运行一个程序，通过运行我们附带的测试程序测试开发板，以及如何擦除flash，如果通过u-boot烧写flash，如何设置u-boot的参数等。
3。学习使用jtag来烧写boot程序，在flash没有u-boot的时候，开发板没有任何程序，超级终端中就没有何人的显示，这时候我们需要使用jtag接口来烧写u-boot程序。然后才可以使用u-boot操作。我们需要熟练的掌握烧写技能。当然flash是有使用寿命的。我们并不建议频繁的烧写flash。
在这个阶段如果学会了这些基本的应用，我们就已经完成了任务。



第二阶段的学习：
在这个阶段的学习难度会增加不少，主要是学习arm编译器ads1.2的使用一及各个功能部分的基本操作。
1.学会使用ads1.2来编译程序，使用附带的jtag+AXD来调试程序
2.认真学习各个功能模块的代码,了解arm的代码特点，我们的test中附带了20多项的功能演示，通过这些演示来学习用程序对各个功能模块的操作，虽然在实际的使用中很少有人让arm脱离操作系统运行。但是这些代码为以后学习arm的驱动程序是很有好处的。对熟悉各个接口的功能也是必不可少的。有的初学者认为比较难，其实这是很基础的代码。需要仔细研究的。入门以后就比较容易接受了。
在这里提醒初学者一点，就是在看这些代码的时候不要为了看而看，需要结合原理图，芯片手册来看，比如控制led的程序，我们首先通过原理图看led接到了那个io口上，然后再通过44b0芯片手册察看这个io的寄存器地址。然后我们察看程序是怎么对这些io口进行控制的。我们学习的目的是独立开发。所以我也希望初学者掌握这种方法。这才是目的。我们的技术支持人员经常谈论说客户反映已经用过几种开发板或者试验箱了，但是除了那个试验箱或者板子，还是对新的项目无从下手。这个的原因就是没有学会分析的方法。



第三阶段：
在这个阶段主要是进行操作系统的学习，也就是最难的学习阶段，目前大部分的嵌入式系统开始采用linux系统，所以我们的开发环境大部分在linux下完成。我们首先要学会使用linux，了解linux的编程环境和方法，同时我们在这里因该学习如何编译u-boot，了解它的结构和移植的方法。接下来我们需要了解uclinux的编译方法和配置环境。记下来就是研究uclinux的移植和驱动程序的编程了。在这个阶段你需要了解liunx的工作机制和makefile等文件的编写。在这个阶段是最漫长的，也是无止境的。
做完了这些。我们已经对arm嵌入式系统的开发已经基本入门了。以后就是根据你的需要学习了。比如使用其它的操作系统，开发项目等等。祝你成功！

第二章使用 U-boot 运行测试程序

第一节设置超级终端

"超级终端"是 windows 自带的一个程序，您可以通过调制解调器、零调制解调器电缆或以太网连接，使用该程序连接到其他计算机、Telnet 站点、公告板系统 (BBS)、联机服务和主机。

尽管将"超级终端"与 BBS 一起使用以访问远程计算机上的信息的方法随着万维网的有效性而变得不再常用，但"超级终端"仍然是配置和测试调制解调器，或检查与其他站点连接的有效方法。

我们在此借助超级终端和串口来调试我们的开发板，在此的超级终端就好像是开发板的输入输出界面了。我们首先来设置超级终端。

首先：[开始]→ [程序]→[附件]→ [通讯]→[超级终端]，如果你是首次使用超级终端，显示如图 2-1 所示

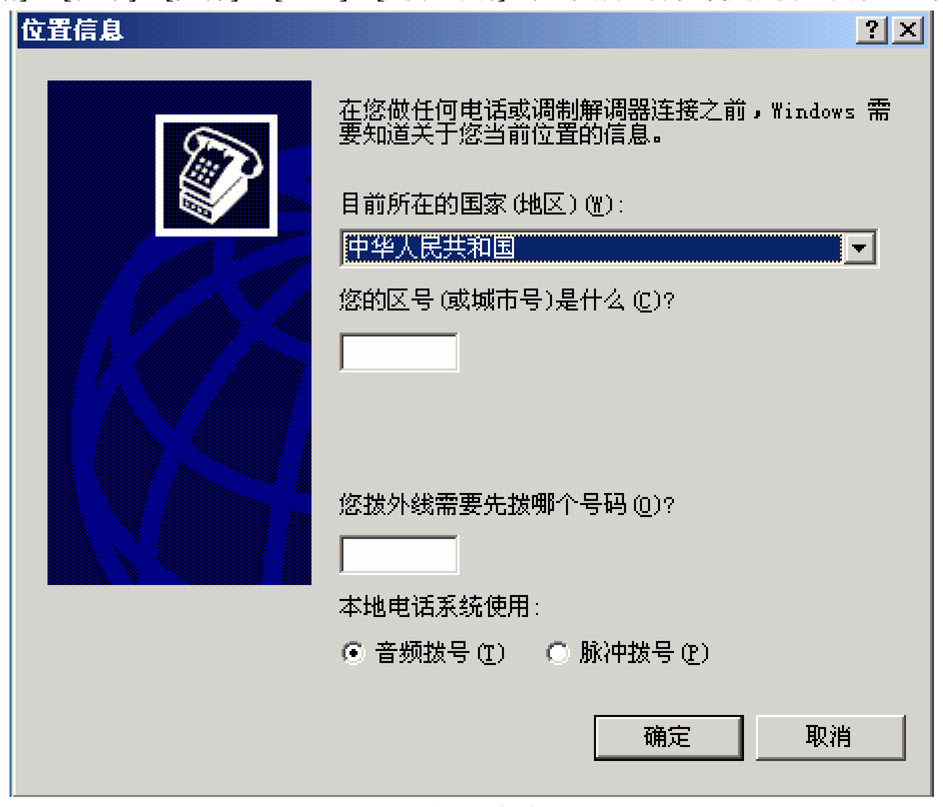


图 2-1 超级终端设置

在区号栏中输入区号：例如 010，点击[确定]，显示如图 1-3，接着点击[确定]，显示如图 2-2 所示

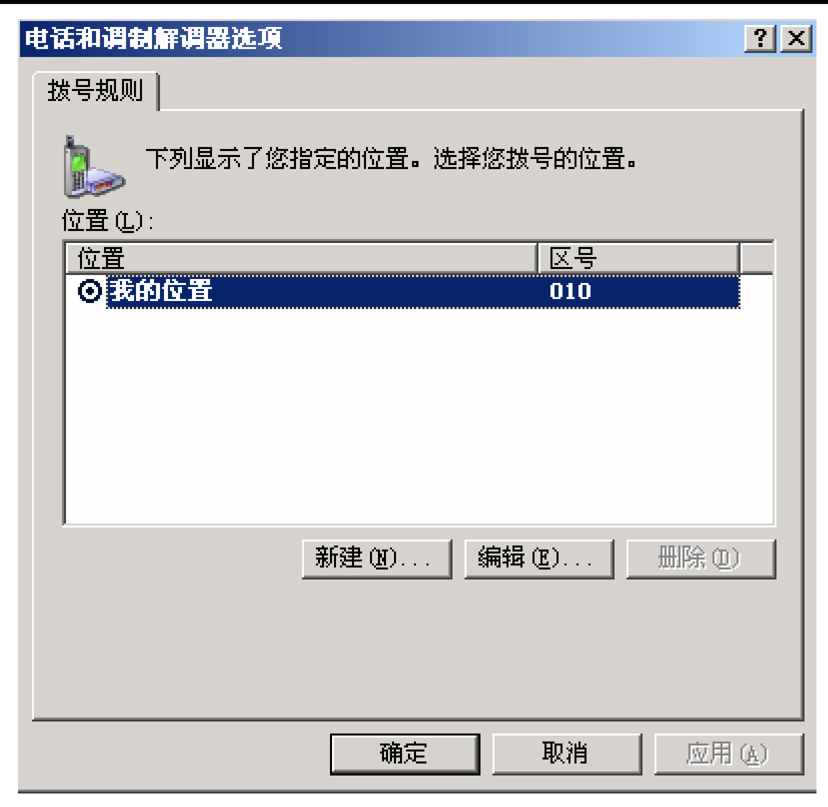


图 2-2 超级终端设置



图 2-3 超级终端设置

在名称栏输入连接的名称，例如 hfrk，点击[确定]，如图 2-4 所示。在连接时使用栏中选择和你的使用相同的串口，你的开发板连接的是 pc 的串口儿，在此就选择 COM 儿。点击[确定]，如图 2-5 所示。

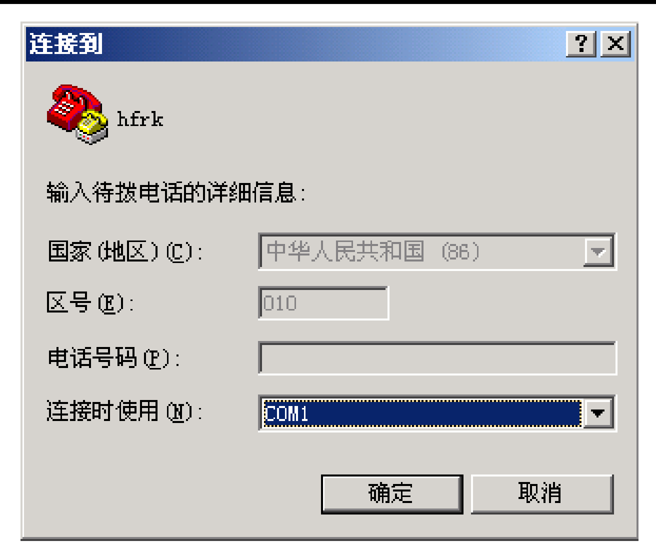


图 2-4 超级终端设置



图 2-5 超级终端设置

设置选择的串口，如图 2-5，速率 115200，数据位 8，无奇偶检验，停止位 1，无流量控制；接下来点击[确定]。超级终端设置完毕。如图 2-6 所示。

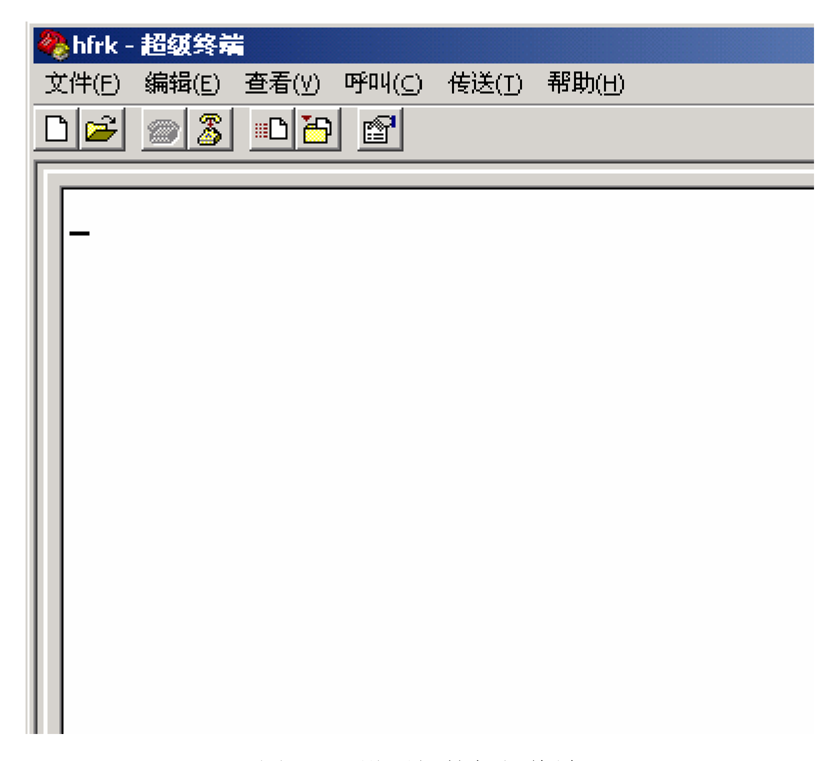


图 2-6 设置好的超级终端

第二节启动 U-boot 程序

将开发板的串口 0 通过串口线(一对一)与 PC 连接，将打开超级终端，打开电源，进入超级终端的启动界面；启动过程中将出现“Hit any key to stop autoboot : 3”由 3 到 0 计数，在计数到 0 之前按回车键，进入 U-boot 命令状态，否则将自动引导 ucliniux。

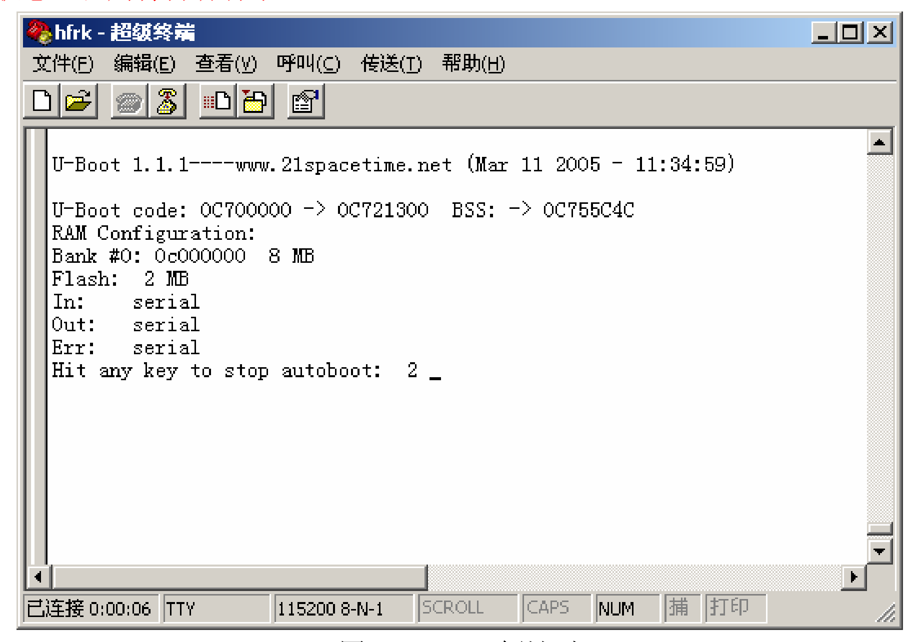


图 2-7 u-boot 倒计时

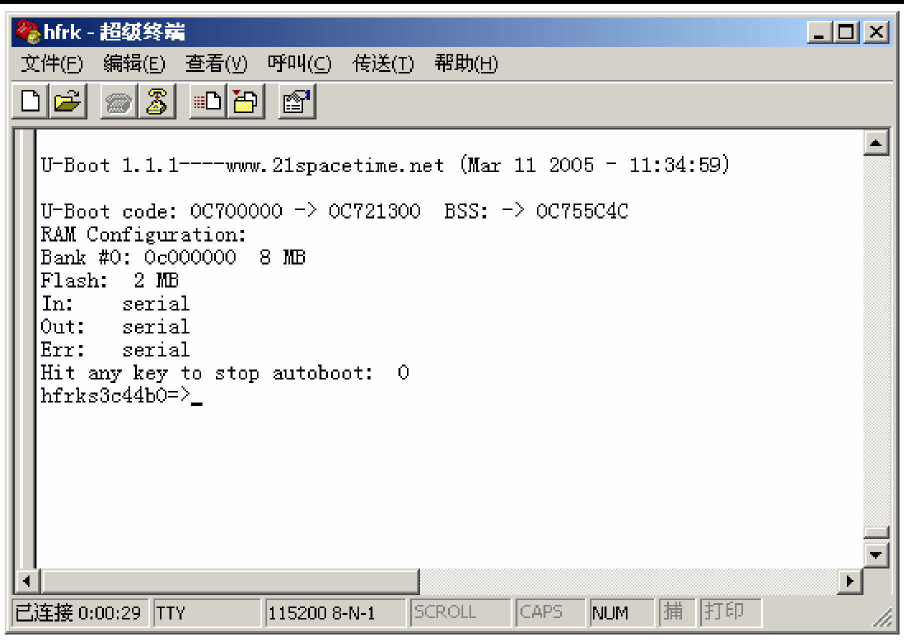


图 2-8 U-BOOT 进入命令状态

注意：有的客户说为什么我看不到 u-boot 的信息，请主要检查串口连线和超级终端的设置（注意流量设置为无，如果设置为软件流量控制或者硬件流量控制，你可能就只能收到信息，不会输入信息，如果开发板的 D1，D2，D3 开机的时候亮一下就灭了，说明 u-boot 已经运行，如果这时候你确定其它的没有问题，但是还看不到信息，请换一台 pc 试试看）。还有好多客户经常把串口线混用，比如用交叉的串口线连接 pc，即使开发板运行，pc 亦不会收到信息。

如果是用 flash 烧完 U-boot 程序后第一次使用，打开电源，超级终端的启动过程中会出现“*** Warning – bad CRC,using default environment”（图 2-3）；输入 save 命令后回车，以后启动 U-boot 时此警告将不会再出现。

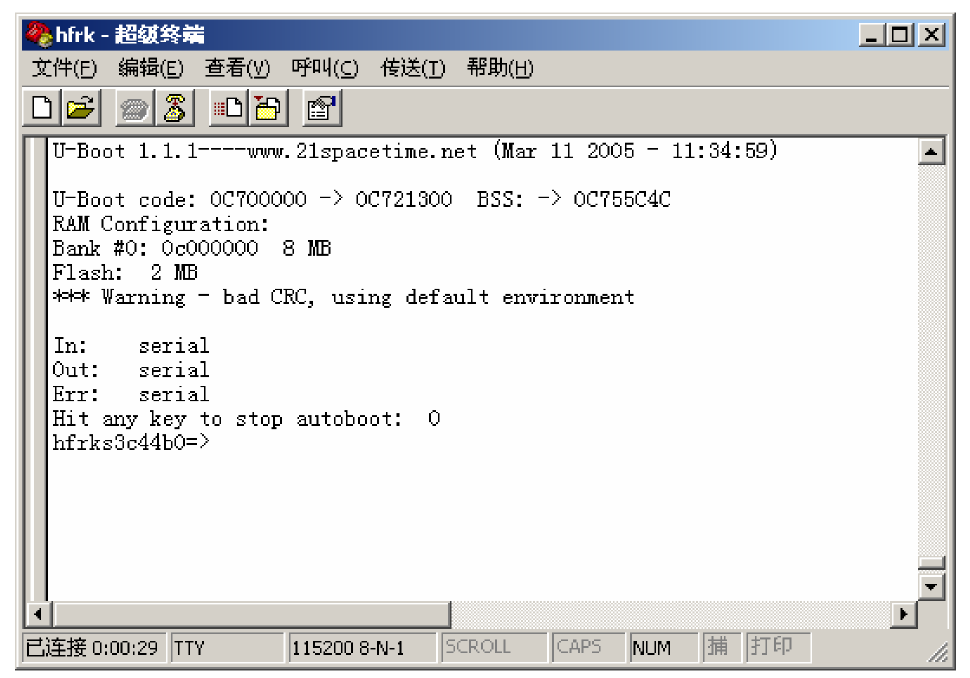


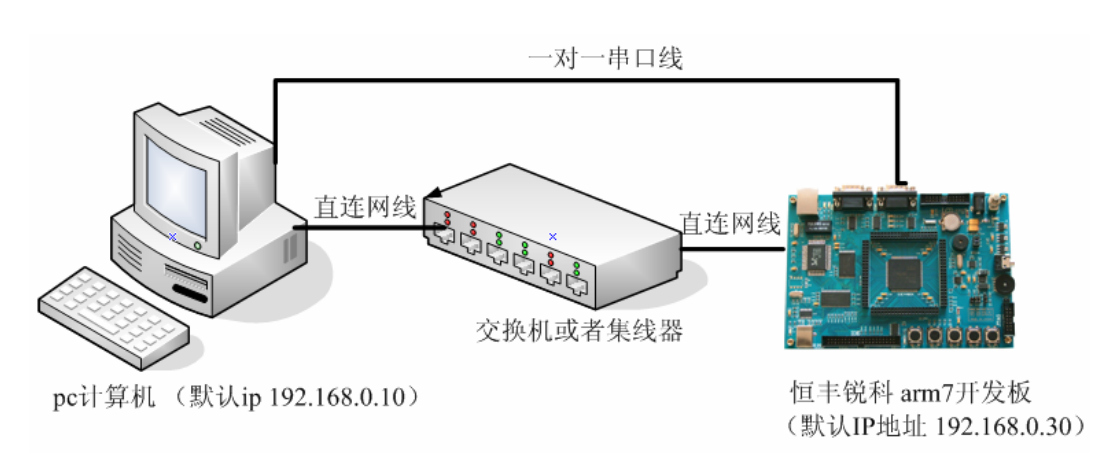
图 2-9 u-boot 出现读取参数 CRC 警告



图 2-10 输入 SAVE 命令保存参数，再次启动警告消失

第三节 u-boot 下载功能

为了运行程序方便我们简单的说明以下下载的原理:



（1）



（2）

图 2-11 tftp 下载原理

如图 2-11（1）所示，是将开发板网线接在了交换机上，（2）是直接和 pc 的网卡连接，它们的原理是一样的，pc 上运行 tftp 的 server 软件，开发板是一个终端设备。它们默认的 ip 地址 pc 是 192.168.0.10，开发板是 192.168.0.30. 需要注意的是两种连接的网线不同，一个是直连的，一个是交叉的网线。

接下来说明如何使用 tftp 下载：首先，打开 tftpd32 这个软件（（工具软件\tftp 目录下），这个目录下还有另一个 tftpserver 软件，同样的功能，用户可以任意选择），tftp32 显示如图 2-2 所示

图 2-12 tftp 服务器软件 tftpd32

在这个软件中我们需要设置的只有 base Directory 这一栏，就是指定一个目录，这个目录下有一需要下载的文件，另外不要设置 server interface 这个选项，这个选项的 ip 地址时自动读取的你的网卡上的 ip，你需要在系统中来设置网卡的 ip，默认是 192.168.0.10。

另外需要指出的是，tftp 下载的文件是参数 bootfile 指定的文件名（我们将在参数设置一节中解释），默认是 u-boot.bin,这个文件必须在 base directory 指定的目录下。

如图 2-13 所示在 u-boot 的命令状态输入 tftp，即可以下载了。

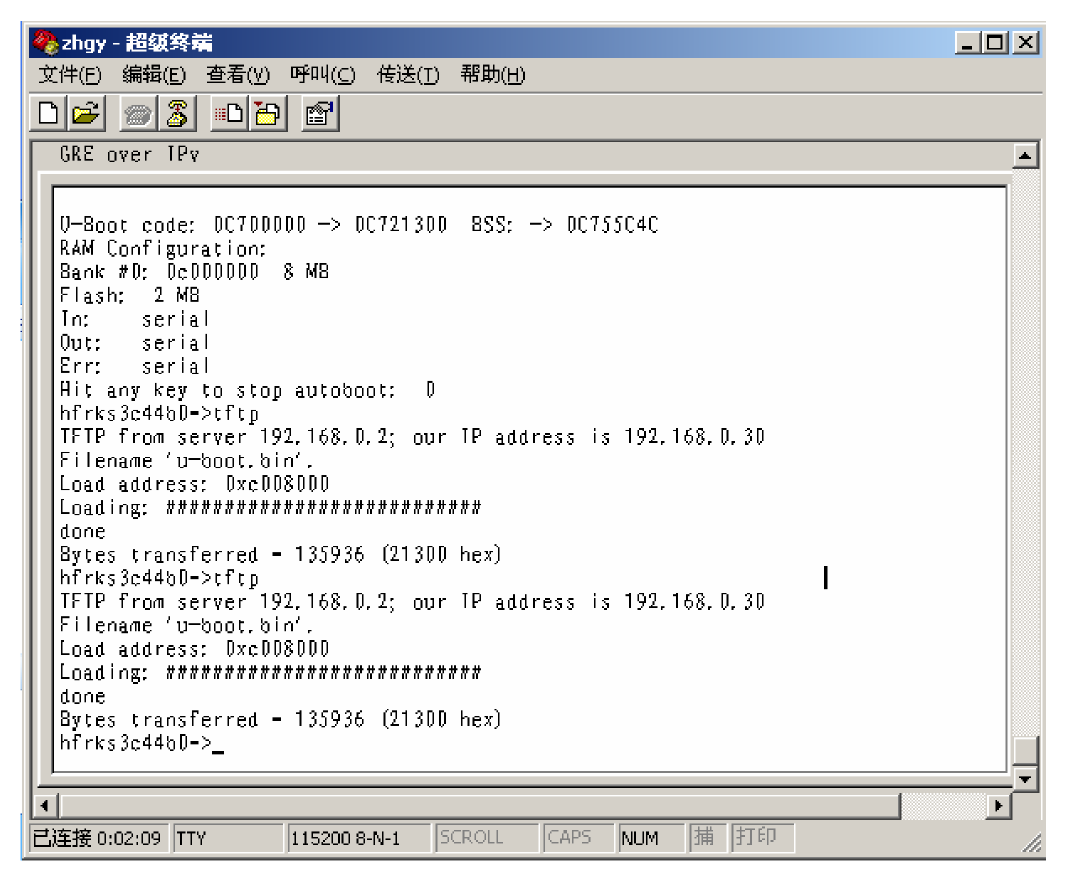
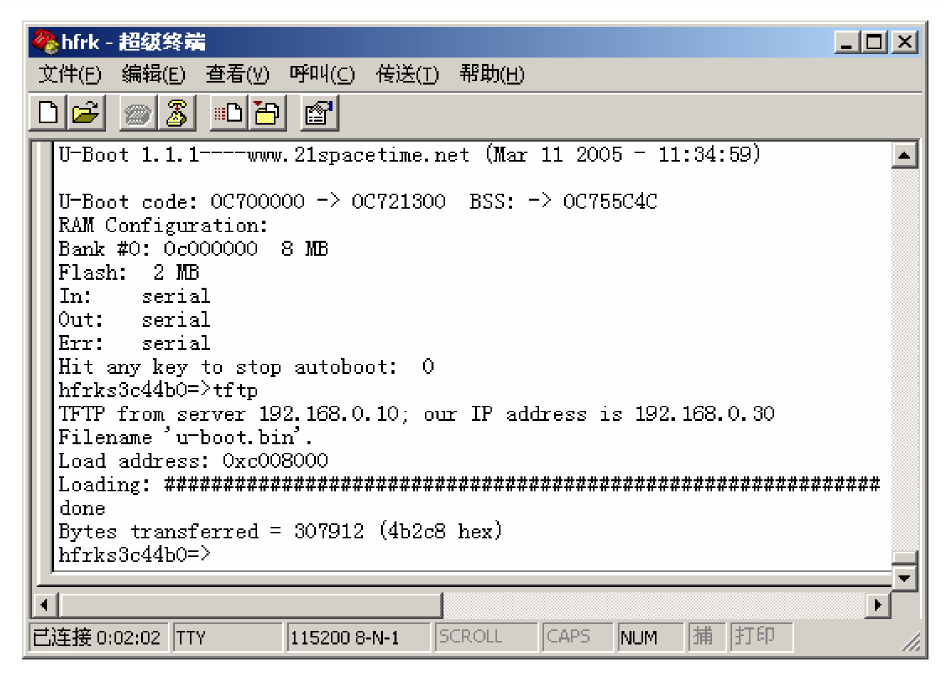


图 2-13 tftp 下载图

输入 tftp 命令	hfrks3c44b0=>tftp
u-boot 指定的参数：	TFTP from server 192.168.0.2; our IP address is 192.168.0.30
	Filename 'u-boot.bin'.
	Load address: 0xc008000
下载进度：	Loading: ###########################
下载完成：	done
下载文件的大小：	Bytes transferred = 135936 (21300 hex)
回到命令状态：	hfrks3c44b0=>

第四节运行测试程序

插入网线，将 LCD 接到开发板的 LCD 端口，拷贝[脱离操作系统的试验代码\综合测试代码\test]目录到你的硬盘，运行 test 目录下的 tftp32，在 U-boot 命令状态输入 tftp 下载 u-boot.bin 文件（在这里这个并非 boot 程序 u-boot，而是和 u-boot 重名的 test.Bin）如图 2-14 所示，然后输入 go 0x0c008000 命令运行刚下载的文件，通过输入不同的命令来测试不同的功能。运行测试文件后，LCD 显示“HengFengRuiKe HF44B0 Test Ver V2.0. . . . . . ” 等字符（图 2-15）。


```
U-Boot 1.1.1----www.21spacetime.net (Mar 11 2005 - 11:34:59)

U-Boot code: 0C700000 -> 0C721300  BSS: -> 0C755C4C
RAM Configuration:
Bank #0: 0c000000  8 MB
Flash:  2 MB
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
hfrks3c44b0=>tftp
TFTP from server 192.168.0.10; our IP address is 192.168.0.30
Filename 'u-boot.bin'.
Load address: 0xc008000
Loading: #############################################################
done
Bytes transferred = 307912 (4b2c8 hex)
hfrks3c44b0=>
```

图 2-14 下载测试程序


```
Load address: 0xc008000
Loading: #############################################################
done
Bytes transferred = 307912 (4b2c8 hex)
hfrks3c44b0=>go 0x0c008000
## Starting application at 0x0C008000 ...?

    欢迎使用恒丰锐科hf44b0 测试程序V2.0 (2007-08-00 )MCLK=60000000

 0:LCD DISPLAY      1:Cache          2:keyboard        3:UART 0
 4:UART 0 FIFO      5:UART 1         6:UART 1 FIFO     7:USB TEST
 8:HOLD Mode        9:Zdma0         10:Zdma1          11:WDTimer
12:RTC(display)    13:RTC(Test)     14:RTC Tick       15:IIC(KS24C08
16:Change PLL      17:Test PWM (BEEP) 18:Test LED     19:UDA1341 Play
20:ADC test

Select the function to test?
```

2-15 运行测试车程序的界面

LCD 测试

在“Select the function to test?”后输入“0”回车，开始测试 LCD：在窗口中显示“Mono test 1.Press any key!”时按回车键（或任意键）LCD 输出黑白相间的正方形格子；再次按回车，LCD 输出一个大正方形（有两条对角线）；继续回车，LCD 输出“HengFengRuiKe HF44B0 Test Ver V2.0……”等字符；最后按回车键退出 LCD 测试。

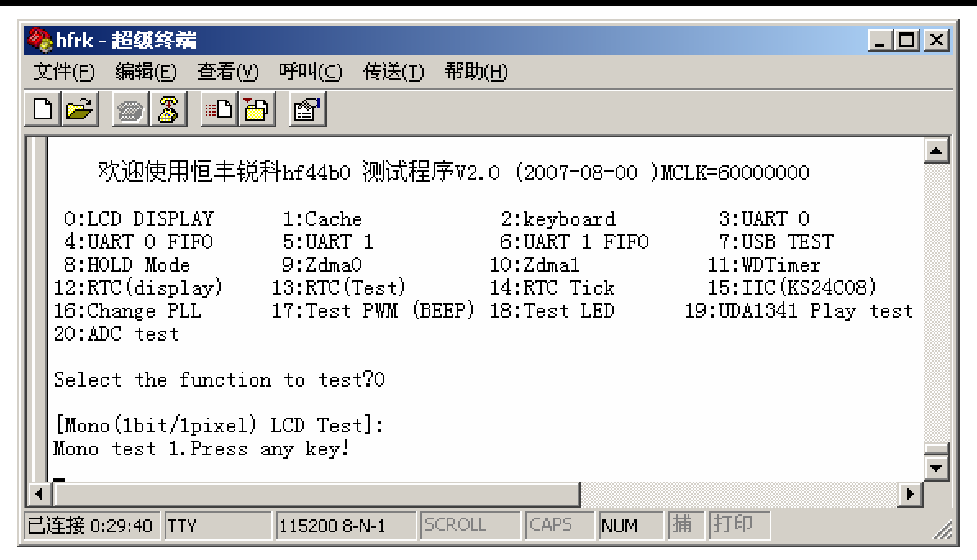


图 2-16 测试 lcd

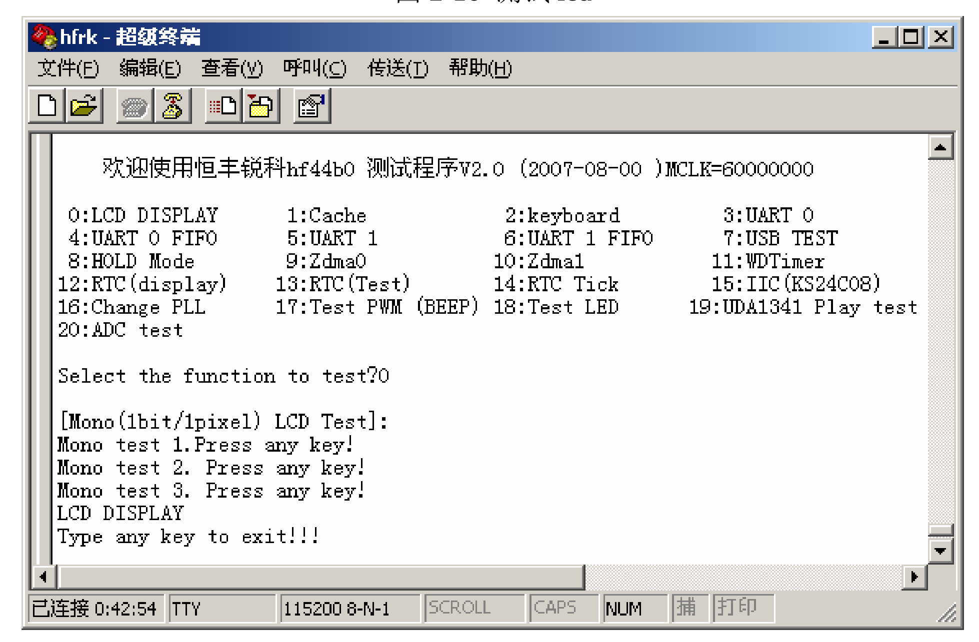


图 2-17 完成 LCD 测试

注意：我们默认的测试程序测试的是 hf34d57 的单色 LCD，如果你测试其它的 lcd 需要跟换测试程序，或者修改测试代码。如果你的 lcd 是 5.7 寸的 hf57A49，你可以使用 LCD 项关目录下的 5.7test.bin 文件测试。

按键测试

在“Select the function to test?”后输入“2”测试 key0~key3 四个按键：如果按下第 1 个键，超级终端窗口和 LCD 均输出“The key is 0”；按下第 2 个键，超级终端窗口和 LCD 均输出“The key is 1”；按下第 3 个键超级终端窗口和 LCD 均输出“The key is 2”；按下第 4 个键超级终端窗口和 LCD 均输出“The key is 3”；说明 4 个按键均正常。

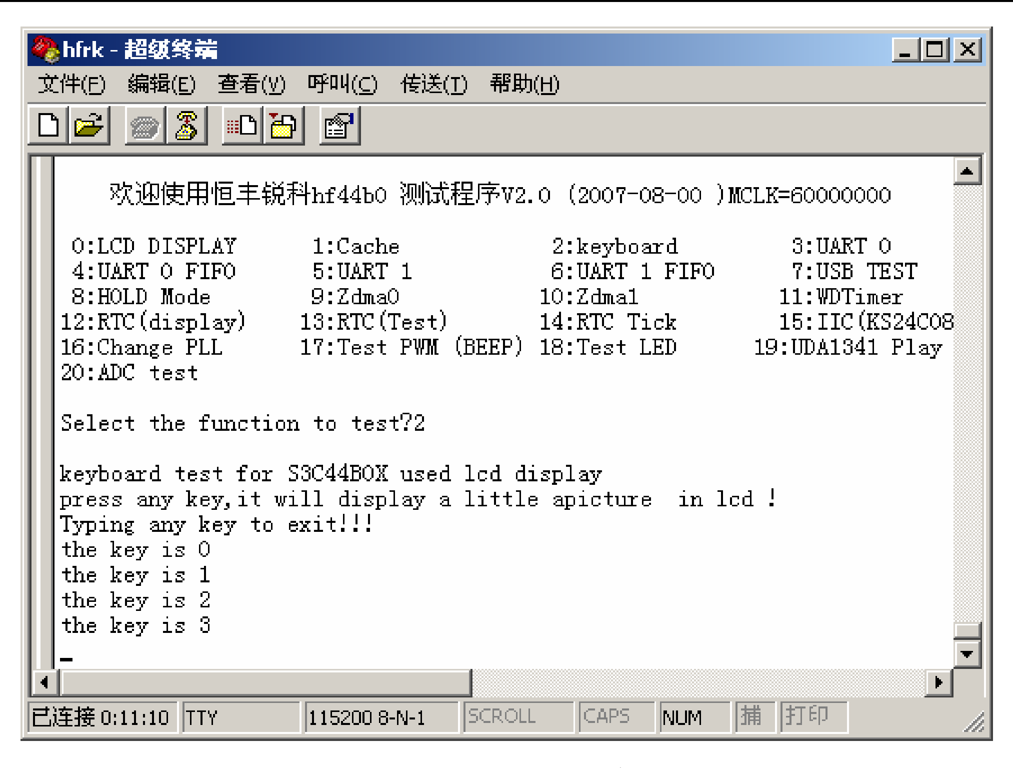


图 2-18 按键的测试程序

串口测试

按键测试完毕按回车键，在“Select the function to test?”后输入“5”测试串口 1：将串口线插入串口 1 后回车，如显示“UART1 Tx Test by BDMA1 is good!!!!”则串口 1 没问题；然后回车，将串口线插回串口 0 后再回车进行其他功能的测试。

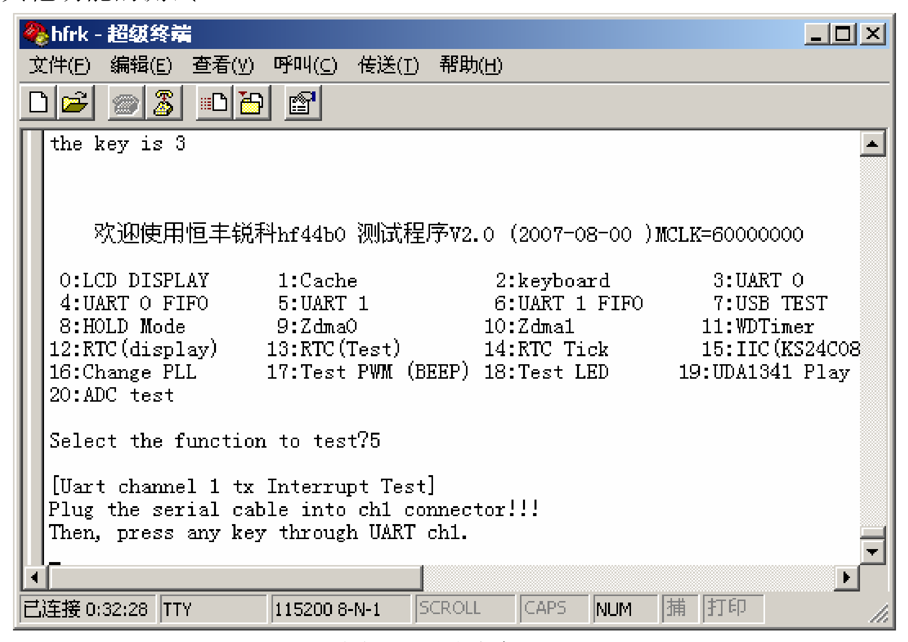


图 2-19 测试串口 1

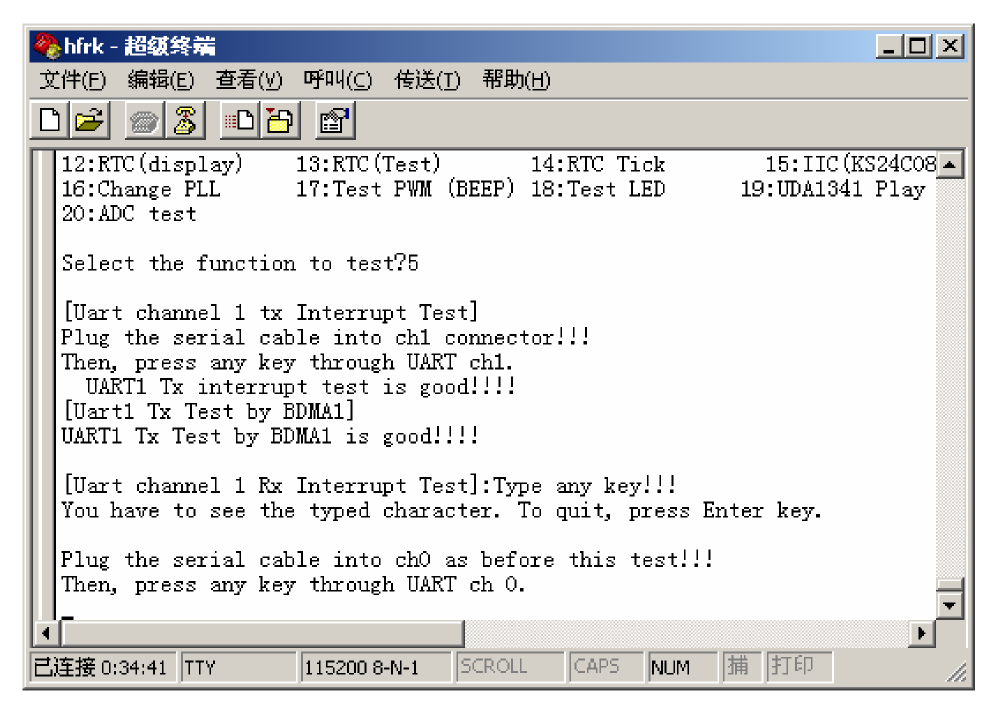


图 2-20 返回串口 0

USB 接口测试

在“Select the function to test?”后输入“7”测试 USB 接口：通过 USB 线将开发板的 USB 接口与 PC 连接，若 USB 接口左侧的指示灯点亮而且计算机右下角的工具栏中显示“发现新硬件”则接口是好的。

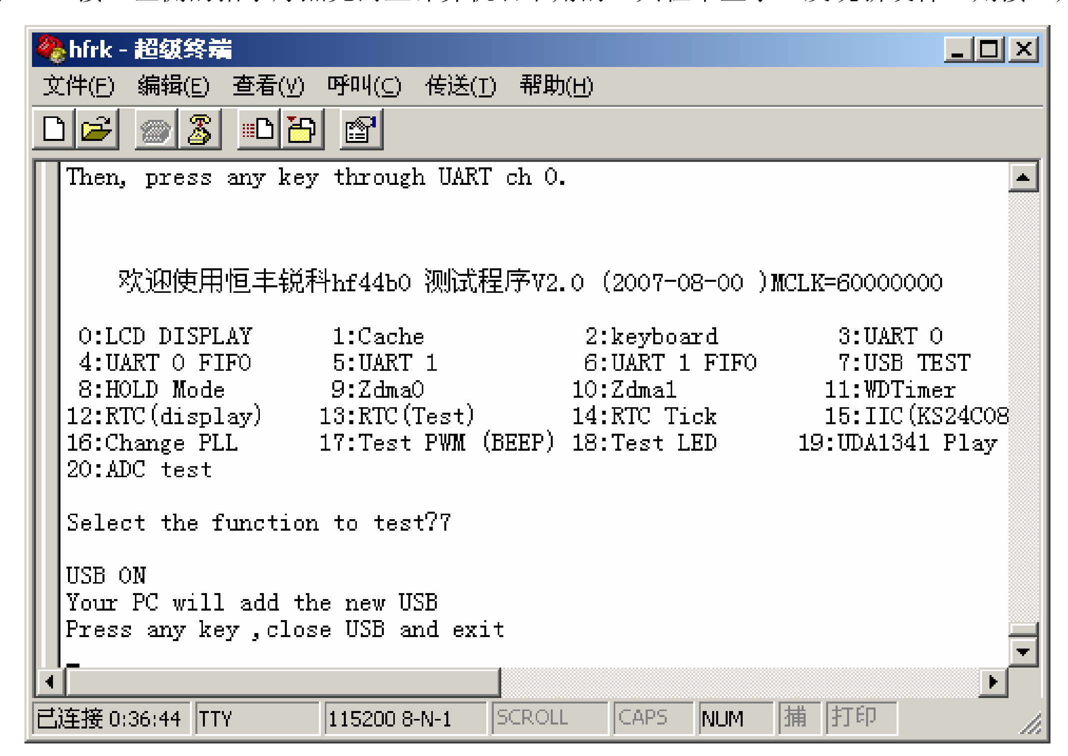


图 2-21 测试 usb 接口

注意：有好多初级客户不时很清楚 usb，usb 分为主机接口和设备接口，主机接口就是 pc 上的接口，可以连接 u 盘等设备。设备接口用来连接主机。44b0 是设备接口，所以只能连接主机，不能连接设备。

系统时钟测试

在“Select the function to test?”后输入“12”通过看时间能否正常运行来判断系统时钟是否正常。

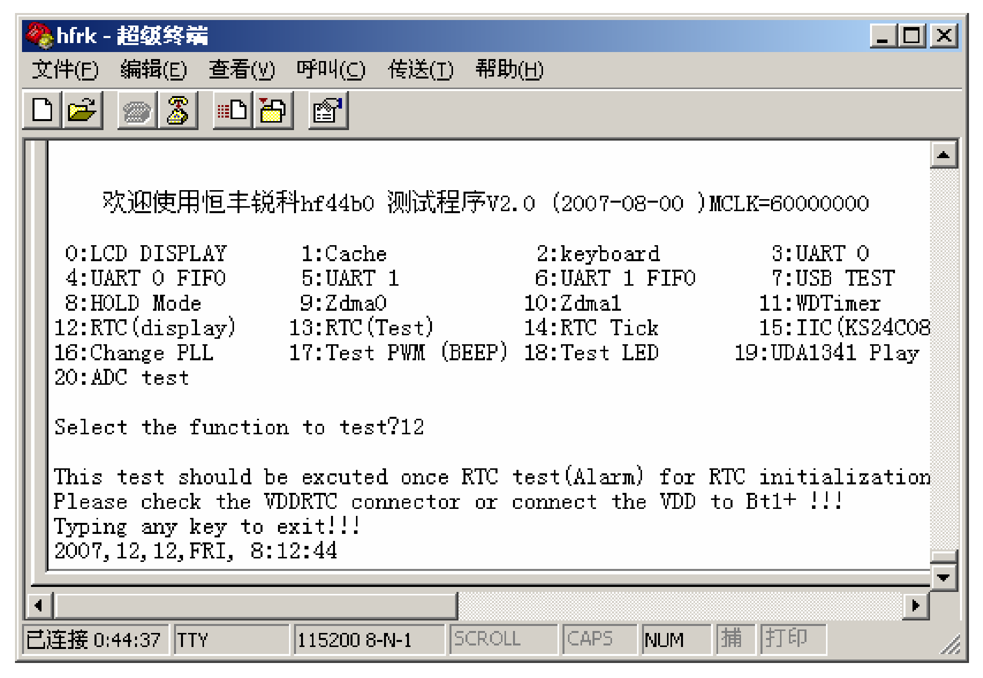


图 2-22RC 测试成功

IIC 总线测试

在“Select the function to test?”后输入“15”回车，若出现下图所示的字符串，则 IIC 总线没问题。

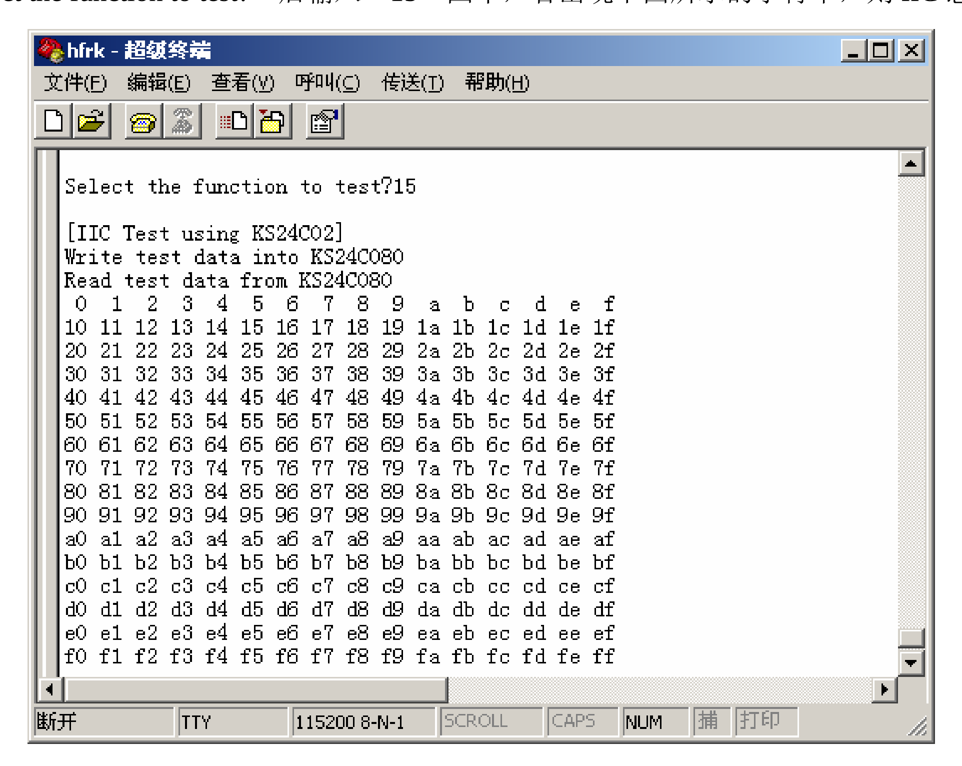


图 2-23 IIC 测试成功

蜂鸣器测试

在“Select the function to test?”后输入“17”回车，然后输入起始和结束参数（在 100--900 之间），若蜂鸣器鸣叫则是好的。

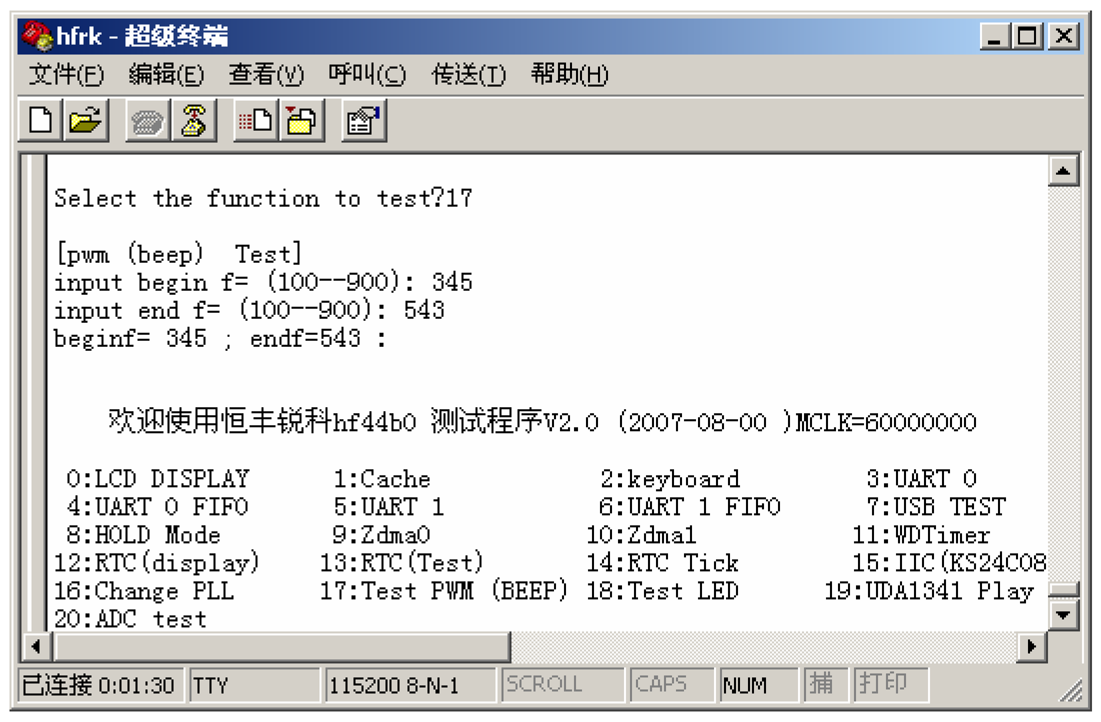


图 2-24 测试蜂鸣器正常

音频测试

在“Select the function to test?”后输入“19”回车，然后将耳机插入开发板上的耳机插座（Speaker），若两只耳塞都有声音则音频输出正常。

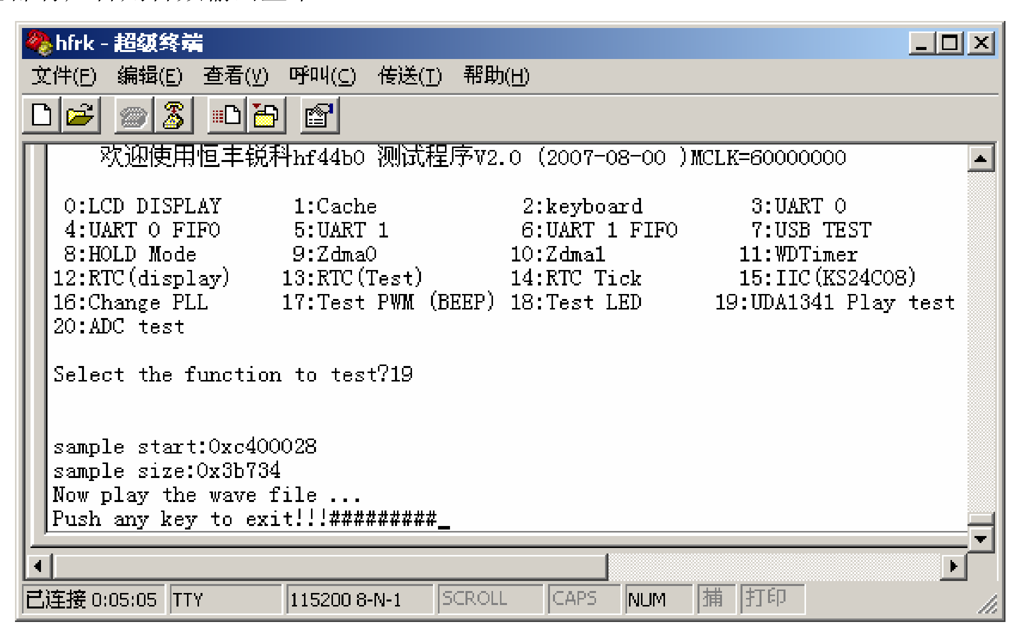


图 2-25 测试播放声音正常

注意：此功能仅限于 4.0 版本

ADC 测试（4.0 版本开发板才可以）

在“Select the function to test?”后输入“20”回车，在 ADIN 端口的 1、2 脚（最右端）插上跳线（连接测试电路和 AD 通道 0）。按下开发板上 KEY0~KEY3 中的任意一个键（一直按着），旋转 ADIN 端口左侧电位器上的旋钮，如果超级终端窗口中的[AIN0]的值随着变化（顺时针旋转变小，逆时针旋转变大）说明 ADC 测试 OK（图 2-27）。

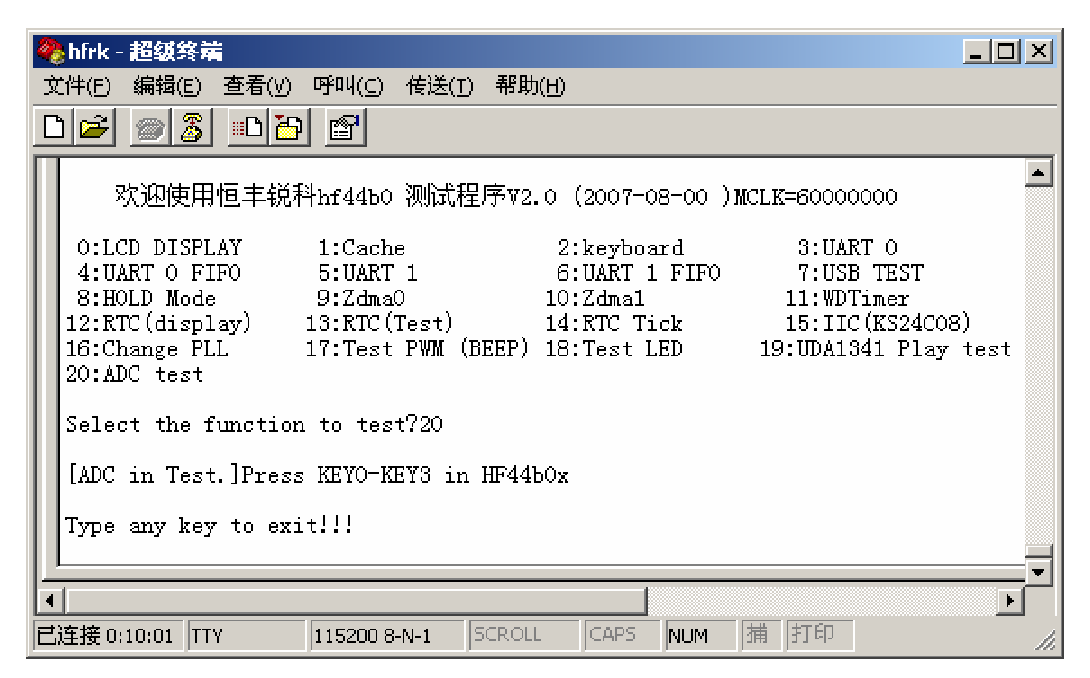


图 2-26 测试 AD 输入

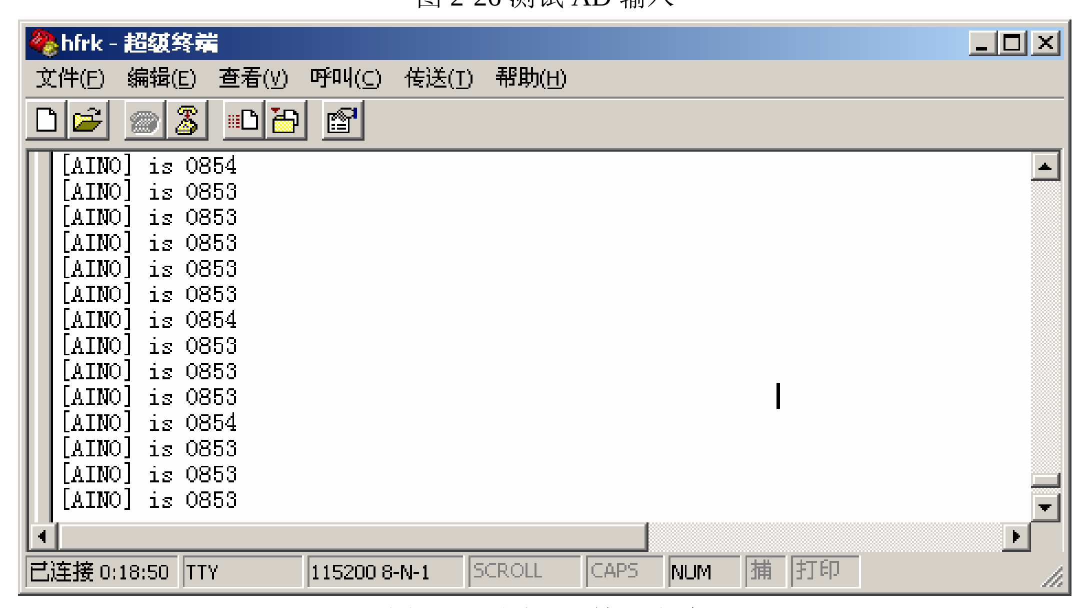


图 2-27 测试 AD 输入成功

第五节用 U-boot 命令烧写 uclinux

首先将开发板的串口 0 通过串口线与 PC 连接，接通开发板电源，插入网线，打开超级终端，打开电源开关，进入 U-boot 命令状态，然后[运行操作系统和 u-boot 源代码\uclinux\uclinux]下的 tftpd32.exe，在 U-boot 命令状态输入 tftp 下载 u-boot.bin（实际为 uclinux 的内核+文件系统，为了方便 tftp 下载修改文件名为 u-boot.bin）文件。

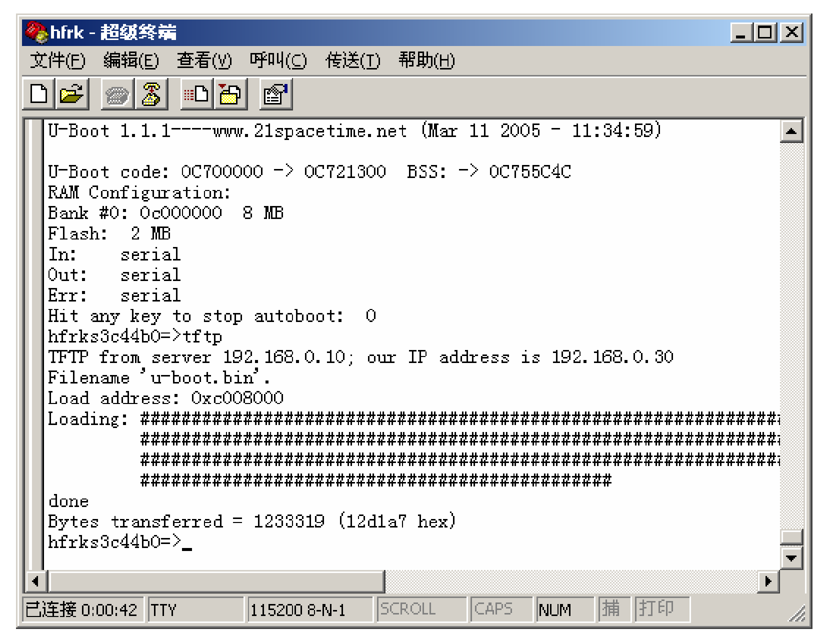


图 2-29 下载完 uclinux 文件

如果你的 flash 不是空的首先要擦除 flash 输入命令

[hfrks3c44b0]erase 0x50000 0x1fffff

等擦除完毕输入命令“cp 0x0c008000 0x50000 4b46b”命令烧写 uclinux：

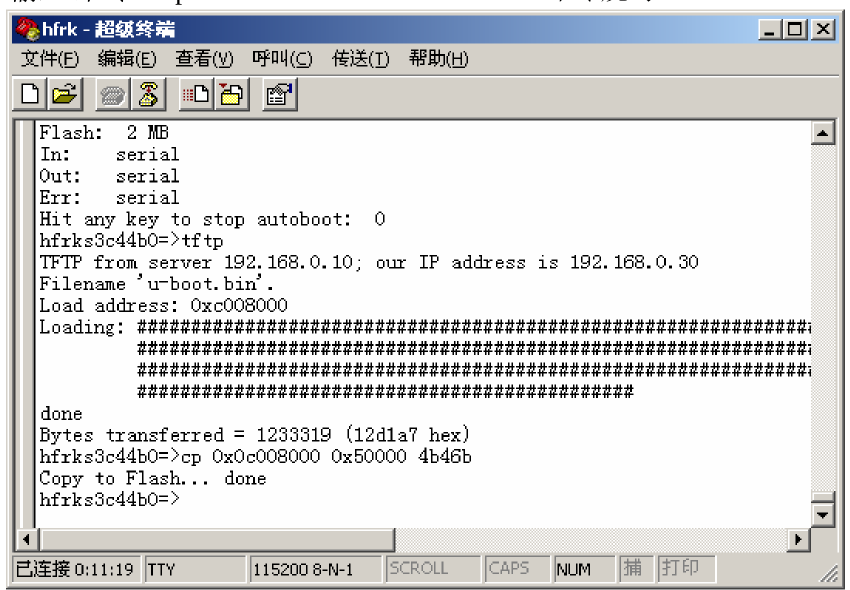


图 2-30 烧写 uclinux

按复位键（RESET）等待系统自动进入 uclinux，如出现下图所示的“uclinux”字样表明烧写成功。

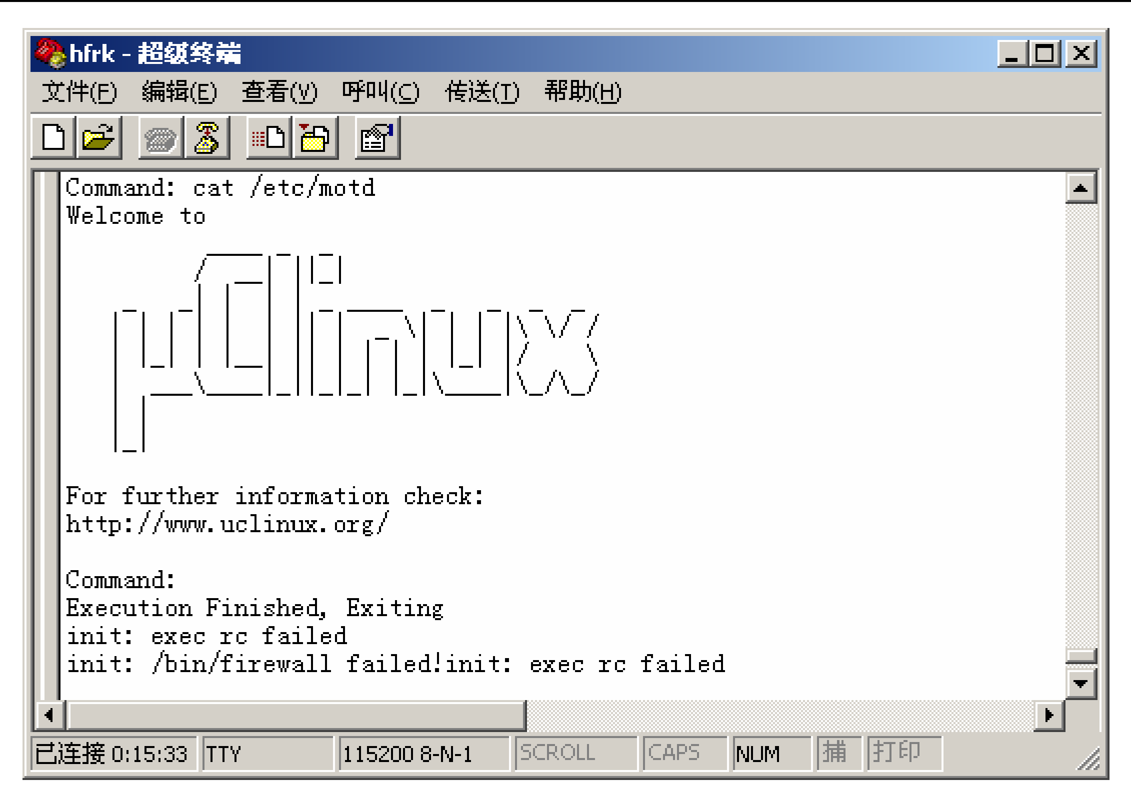


图 **2-31uclinux** 启动后的界面

第三章 U-boot 使用说明

在上一章我们介绍了使用串口下载一个简单的程序并运行的过程，在这一章我们进一步说明 u-boot 的使用。

第一节下载功能

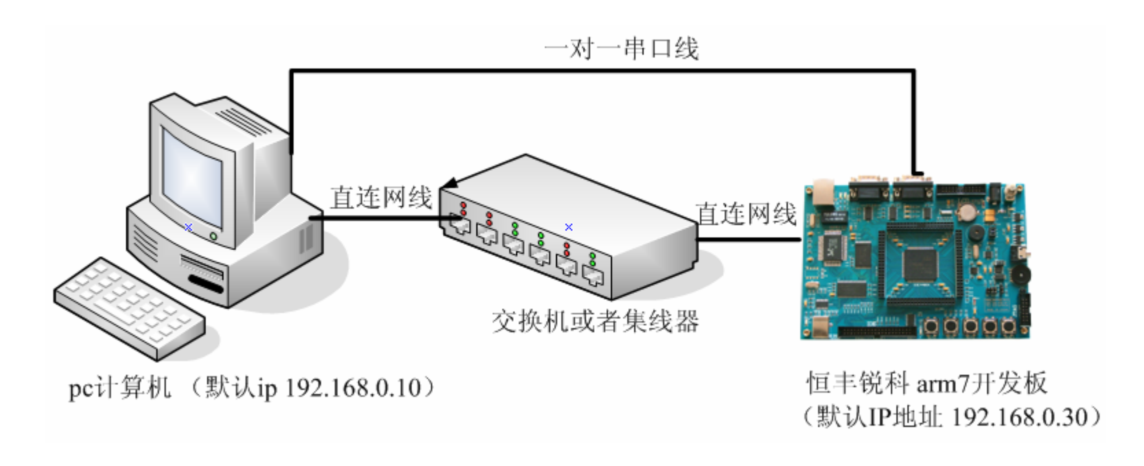
我们已经初步知道了下载的命令 loadb，和运行的命令 go ,我们进一步说明其他常用的命令。

命令	功能
Loadb	用于串口下载文件，使用 kermit 协议
Tftp	使用网口下载文件，使用 tftp 协议
Go	运行.bin 文件
Bootm	运行 u-boot 特定的执行文件，解压以后运行，uclinux_rom 就是这种格式。

我们在上一章中简单的介绍了 loadb 命令，我们接下来介绍一下网口下载的命令 tftp,命令格式：

> tftp [地址]　　如果直接输入 tftp 默认的下载地址是 0x0c008000

下载原理如图 3-1。



（1）

一对一串口线

交叉网线

pc计算机 （默认ip 192.168.0.10）

恒丰锐科 arm7开发板
（默认IP地址 192.168.0.30）

（2）

图 3-1 tftp 下载原理

如图 3-1（1）所示，是将开发板网线接在了交换机上，（2）是直接和 pc 的网卡连接，它们的原理是一样的，pc 上运行 tftp 的 server 软件，开发板是一个终端设备。它们默认的 ip 地址 pc 是 192.168.0.10，开发板是 192.168.0.30. 需要注意的是两种连接的网线不同，一个是直连的，一个是交叉的网线。

这个些 ip 地址可以修改，我们将在参数设置一节中解释。

接下来说明如何使用 tftp 下载：首先，打开 tftpd32 这个软件（工具软件\tftp，这个目录下还有另一个 tftpserver 软件，同样的功能，用户可以任意选择），tftp32 显示如图 2-2 所示

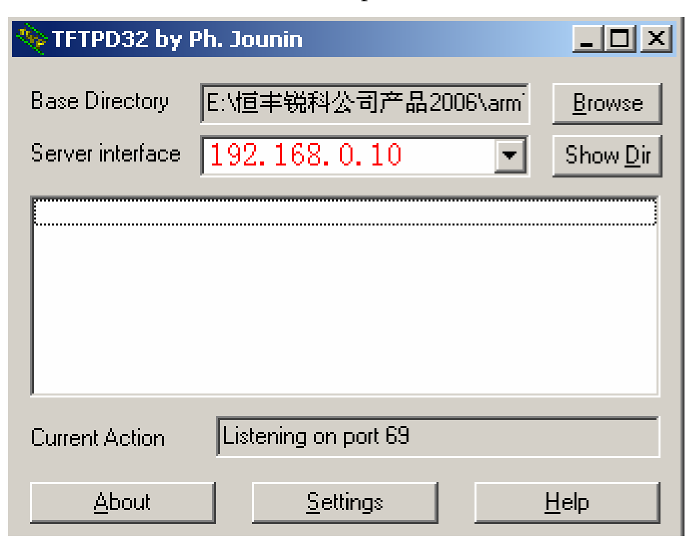


图 3-2 tftp 服务器软件 tftpd32

在这个软件中我们需要设置的只有 base Directory 这一栏，就是指定一个目录，这个目录下有一需要下载的文件，另外不要设置 server interface 这个选项，这个选项的 ip 地址时自动读取的你的网卡上的 ip，你需要在系统中来设置网卡的 ip，默认是 192.168.0.10。

另外需要指出的是，tftp 下载的文件是参数 bootfile 指定的文件名（我们将在参数设置一节中解释），默认是 u-boot.bin,这个文件必须在 base directory 指定的目录下。

如图 3-3 所示在 u-boot 的命令状态输入 tftp，即可以下载了。

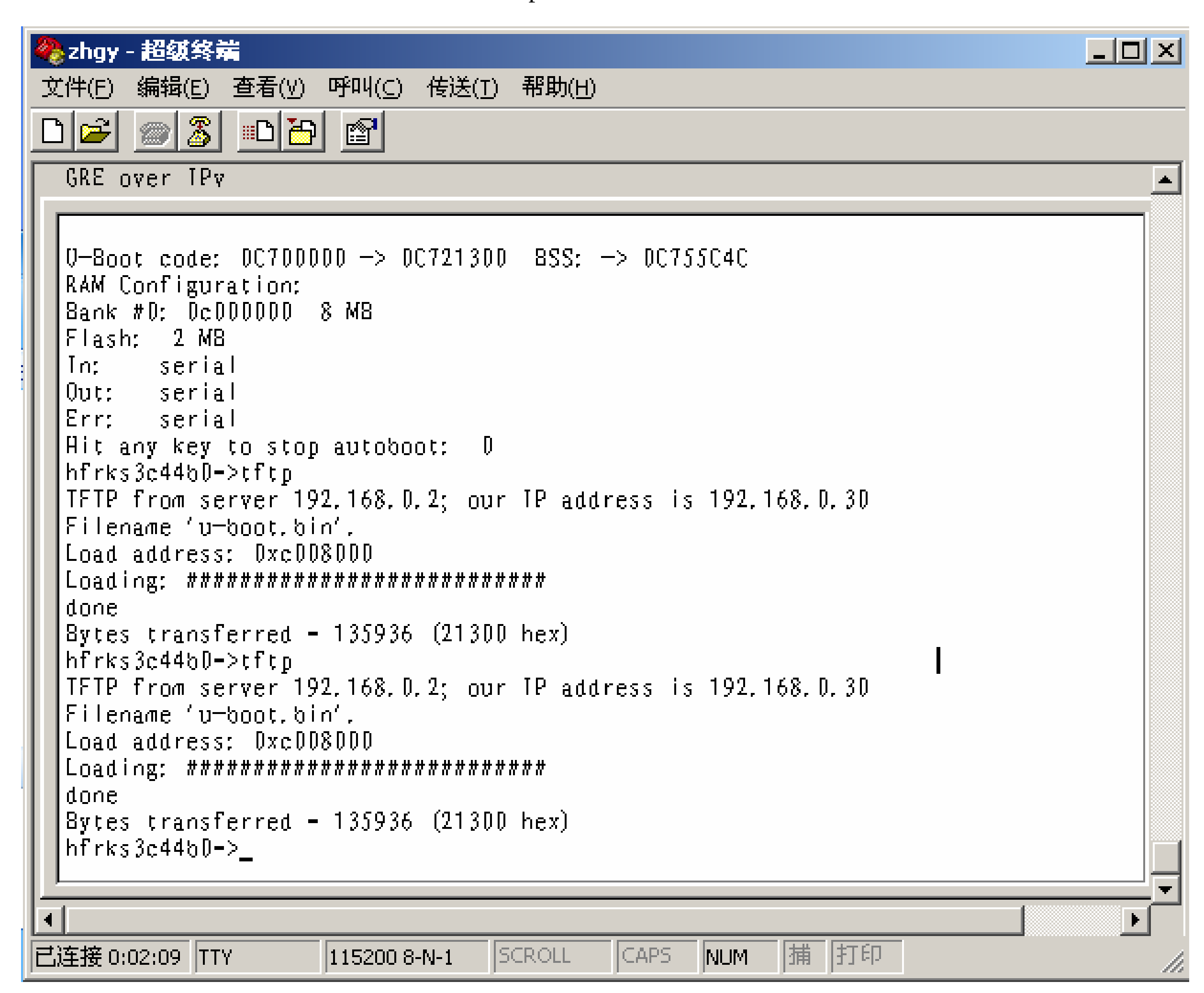


图 3-3 tftp 下载图

输入 tftp 命令	hfrks3c44b0=>tftp
u-boot 指定的参数：	TFTP from server 192.168.0.2; our IP address is 192.168.0.30
	Filename 'u-boot.bin'.
	Load address: 0xc008000
下载进度：	Loading: ############################
下载完成：	done
下载文件的大小：	Bytes transferred = 135936 (21300 hex)
回到命令状态：	hfrks3c44b0=>

现在我们可以使用 go 0x0c008000 来运行这个下载的文件了。需要注意的是：如果我们下载的时 uclinux_rom.bin 这文件，因为它是压缩格式，所以需要使用 bootm 启动运行。Bootm 是首先解压到 0x0c008000 位置，然后运行的。所以我们下载压缩格式就不能下载到 0x0c008000 这个位置了，否则将自我覆盖。我们可以下载到 0xc500000 的位置。命令是：tftp 0x0c500000

第二节 u-boot 参数的设置

我们在超级终端中输入 print，将显示 u-boot 的所有参数，如图 2-4 所示。

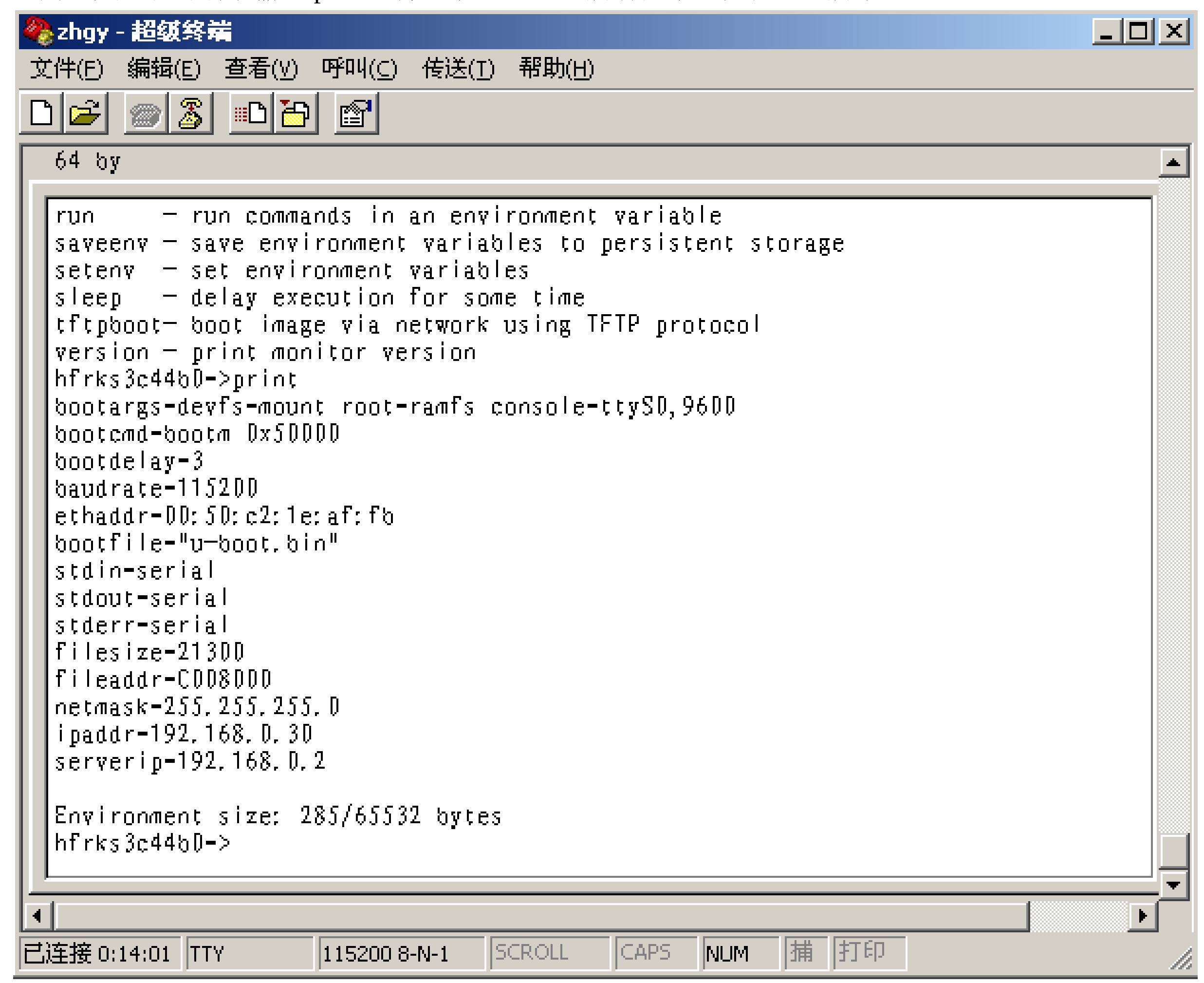


图 3-4 u-boot 显示参数

主要的参数如下：

bootcmd=bootm 0x50000	注：在启动的过程中将出现“Hit any key to stop autoboot: 3”其中数字由 3 到 0 计数，如果回车进入命令状态，如果不回车，执行的就是这个参数设置的命令，默认是 bootm 0x50000,也就是启动 0x50000 位置的压缩内核。也就是我们预装的 uclinux。
bootdelay=3	注：在启动的过程中将出现“Hit any key to stop autoboot: 3”，其中 3 这个数字就是这个参数指定的。
baudrate=115200	注：串口速率
ethaddr=00:50:c2:1e:af:fb	注：以太网物理地址
bootfile="u-boot.bin"	注：tftp 下载的文件名
ipaddr=192.168.0.30	注：开发板的 ip 地址
serverip=192.168.0.10	注：tftp 下载时，pc 的地址。

这些参数都是可以修改的，保存在 flash 的 0x40000 到 0x4ffff 之间。修改的命令是 setenv（可以简写为 set），命令格式是

Set 参数名称 参数

修改完成以后需要保存这个参数，命令是 save

例如我们想把 serverip 这个参数修改为 192.168.0.2

输入：

hfrks3c44b0=>set serverip 192.168.0.2

hfrks3c44b0=>save

我们重新输入 print 查看这个参数，已经修改为了 192.168.0.2。其他的参数了是类似。

我们需要注意以下 bootcmd 这个参数，我们已经知道这个参数是自动启动以后执行的命令，那么能不能执行 1 个以上的命令呢，答案是肯定的。如我们需要自动执行一个程序，程序放在了 0x50000 的位置，我们可以这样来运行，首先拷贝这个程序到 sdram 的 0x0c008000 位置，然后执行 go 运行，可以这样设置这个命令（假设拷贝长度是 4d4d）

hfrks3c44b0=>set bootcmd cp 0x50000 0x0c008000 4d4d\; go 0x0c008000

hfrks3c44b0=>save

在此可以明确 2 个命令使用\;隔开就可以了。

第三节使用 u-boot 来烧写 flash

使用 boot 的功能更大的好处是使下载烧写程序更加快捷方便。在线更新程序更加容易，u-boot 对 flash 的操作主要是写入和擦除。

命令	描述
cp	拷贝内容，同时可以烧写 flash
erase	擦除 flash，分块擦除

在这里我们需要说明的是 flash 的地址，我们的开发板一共 2M 的 flash，地址分配如下：

0x0-0x3ffff　　u-boot 的代码区

0x40000-0x4ffff　　u-boot 的参数区

0x50000-0x1fffff　　用户程序区

首先介绍 Cp 命令的使用，这个命令用于拷贝文件，格式是：

Cp 源地址 目标地址 长度

比如我们下载了一个文件，如下所示，我们知道我们下载到了 0x0c008000 的位置，我们想把这个文件烧写到 flash 的 0x50000 位置，怎么办呢？

```
hfrks3c44b0=>tftp
TFTP from server 192.168.0.2; our IP address is 192.168.0.30
Filename 'test.bin'.
Load address: 0xc008000
Loading: ############################
done
Bytes transferred = 135936 (21300 hex)
hfrks3c44b0=>
```

首先计算需要拷贝的长度，21300/4+2=84c2 我们的命令就是：

```
hfrks3c44b0=>cp 0x0c008000 0x50000 84c2
```

第四节编译 u-boot

恒丰锐科采用的 bootloader 程序编译

第一：首先在 linux（或其模拟环境）解压源文件

命令： tar -zxvf u-boot-xxxxxxxx.tar.gz

第二：配置你的开发板

命令：make mrproper
make hfrks3c44b0_config

第三：编译程序

命令：make

第四：烧写代码

烧写:u-boot.bin 文件到你的开发板

相关文件以及目录：

1.> \u-boot\cpu\s3c44b0 cpu 相关代码

2.> \u-boot\board\hfrk 板级支持代码

3.> \u-boot\include\configs\hfrks3c44b0.h 开发板配置文件

注：我们的源代码是 make 以后的所有部分，所以你可以直接 make,你还可以用 make mrproper 清楚在重新编译。

第四章 ADS1.2 的安装和使用

第一节 ADS1.2 的安装过程

首先解压 ads12setup.zip 文件（在[\windows 下的编译器]下面），然后双击此文件夹下的 SETUP.EXE 程序开始安装。

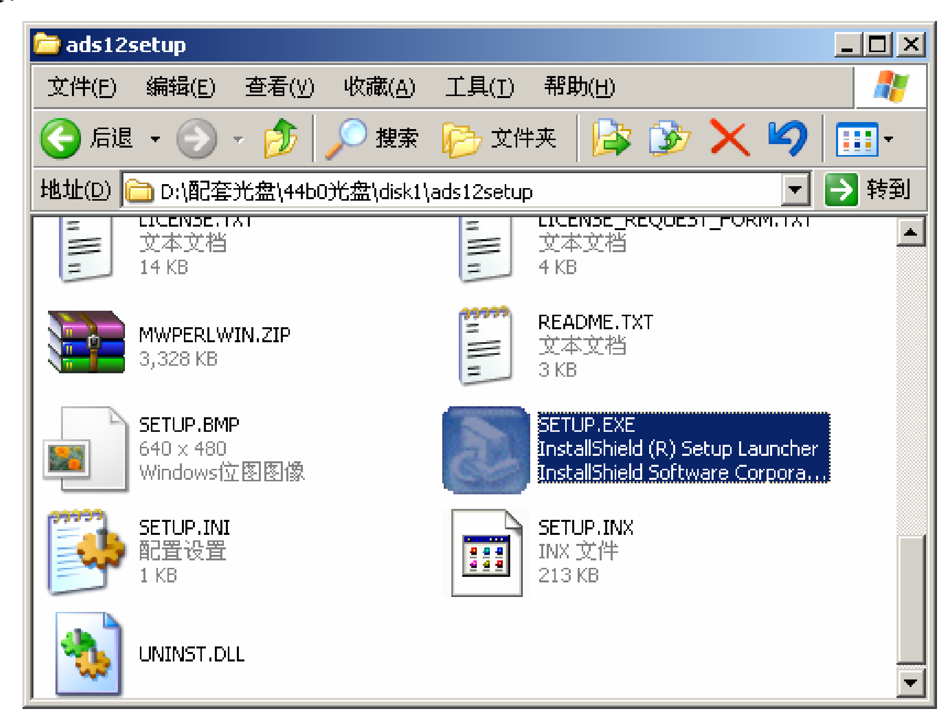


图 4-1 运行 SETUP.EXE

单击“Next”进入下一步：

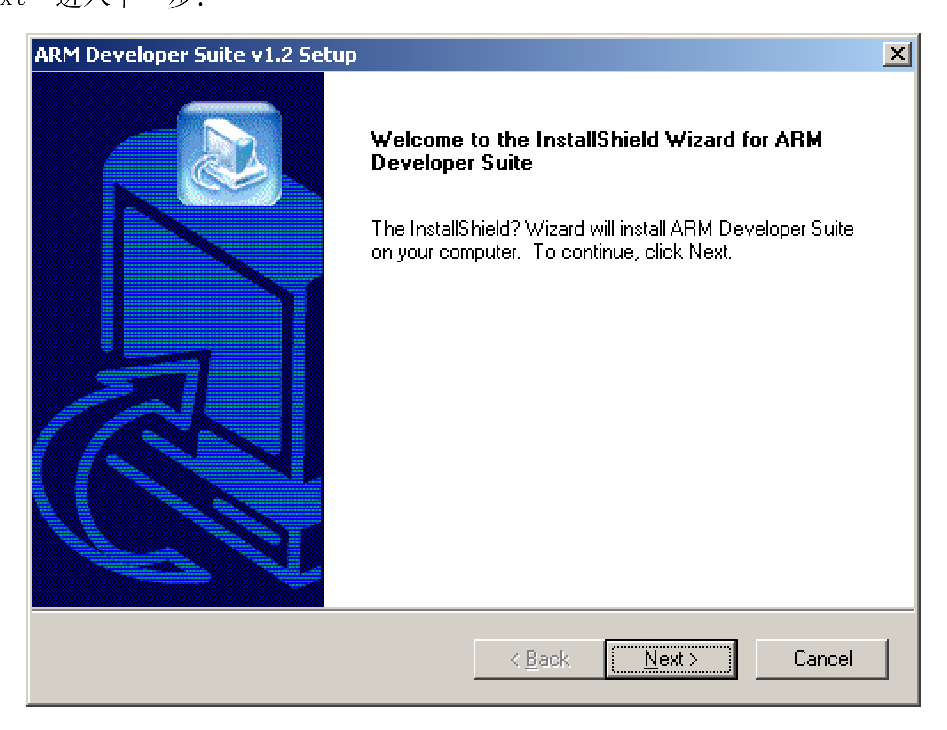


图 4-2 ads1.2 安装第 2 步

单击“Yes”继续：

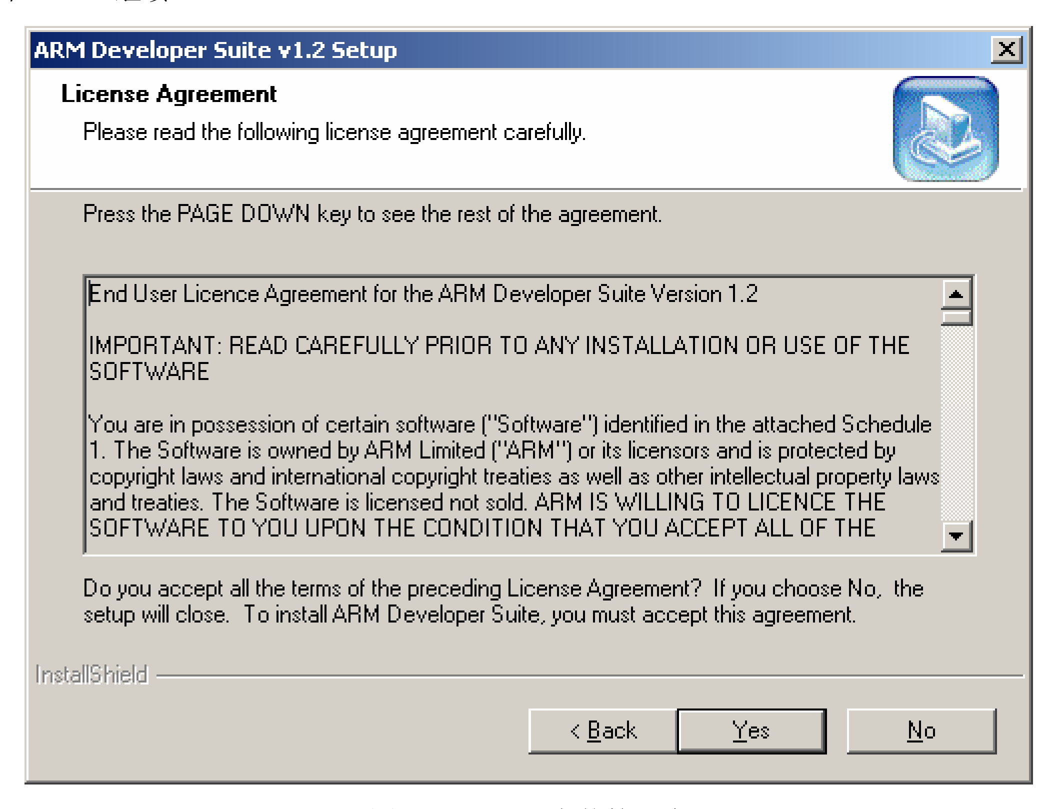


图 4-3 ads1.2 安装第 3 步

默认的目标文件夹为 C:\ Progam Files\ARM\ADSv1_2（也可以单击 Browse 自定义目标文件夹），单击“Next”进入下一步：

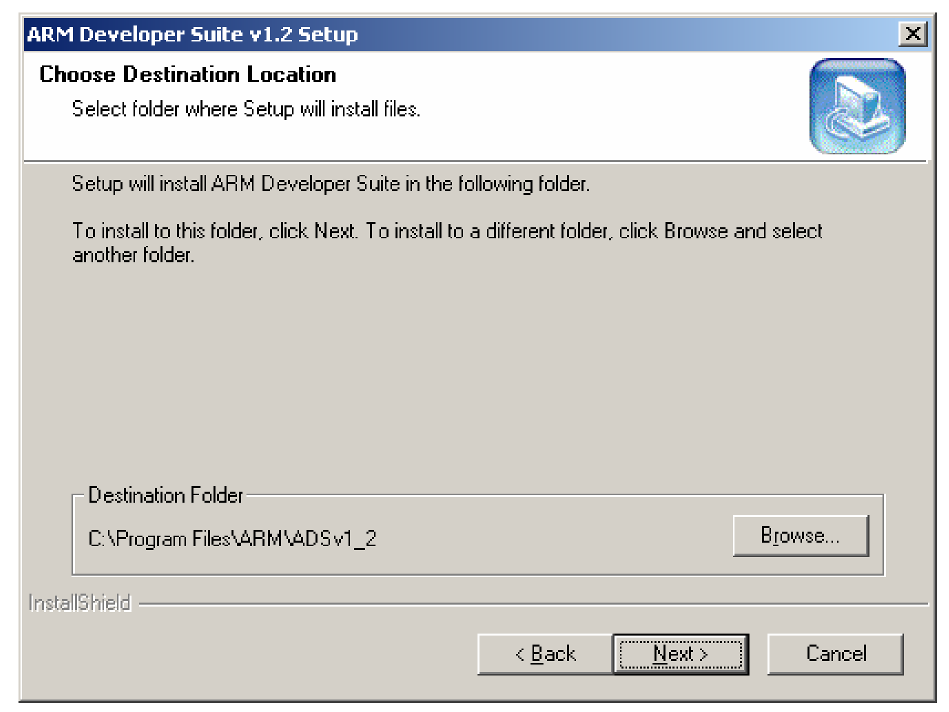


图 4-4 ads1.2 安装第 4 步

在“Click the type of Setup you prefer”中选择“Full”：

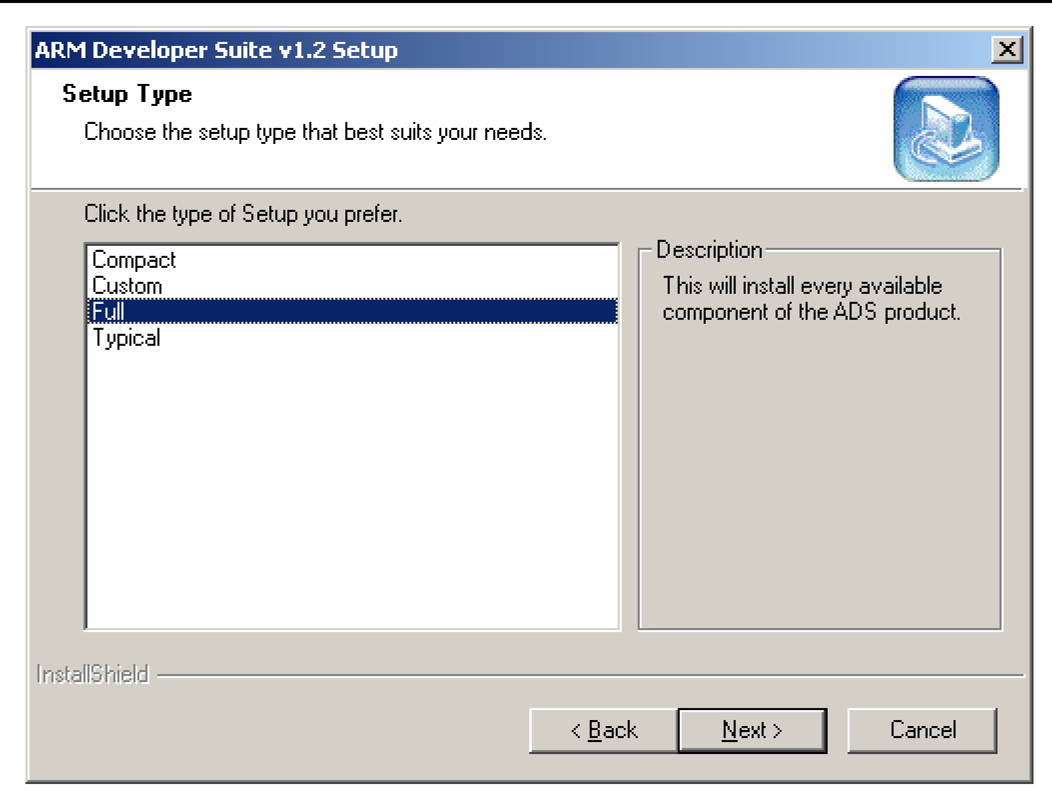


图 4-5 ads1.2 安装第 5 步

在 Program Folder：中可以自己命名程序文件夹，默认的是 ARM Developer Suite v1.2：



图 4-6 ads1.2 安装第 6 步

选择相关文件时全选即复选框内全部打勾：

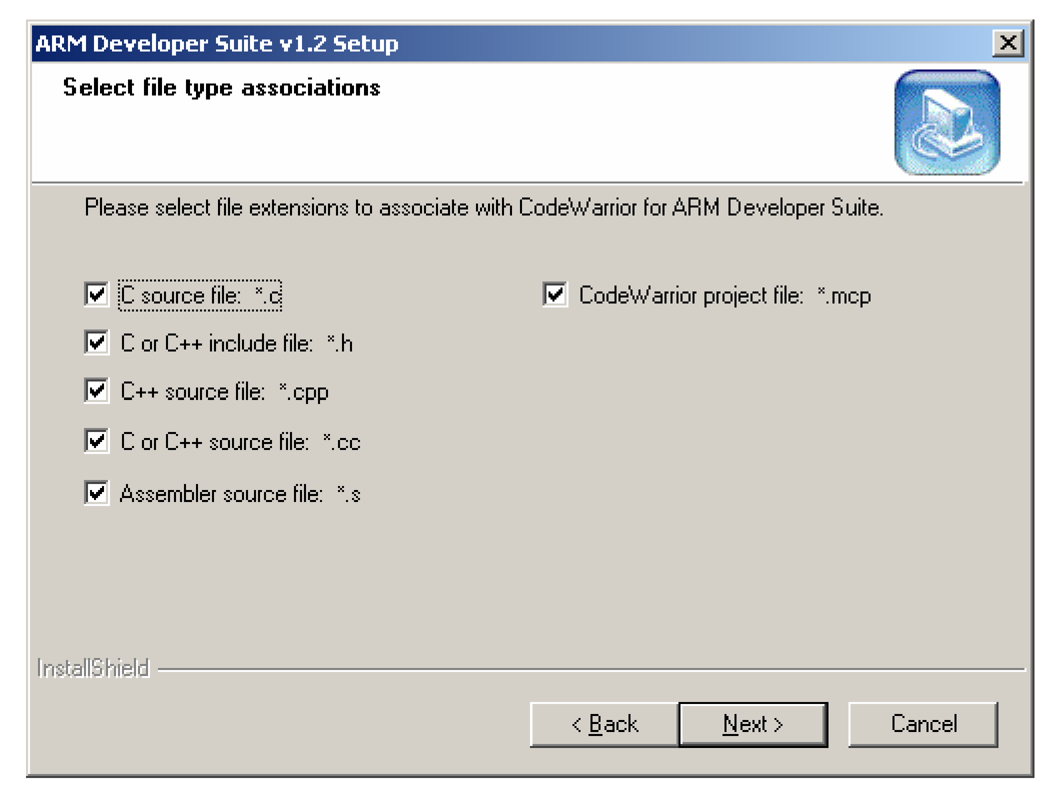


图 4-7 ads1.2 安装第 7 步

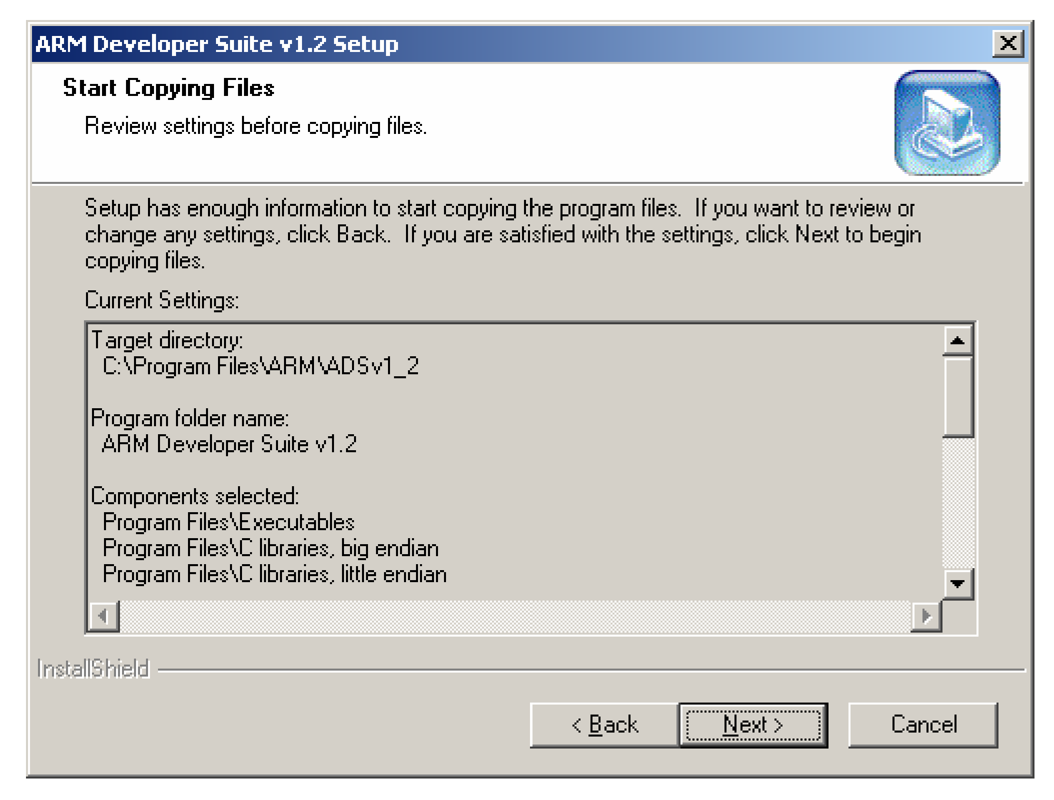


图 4-8 ads1.2 安装第 8 步

“Next”后开始复制文件：

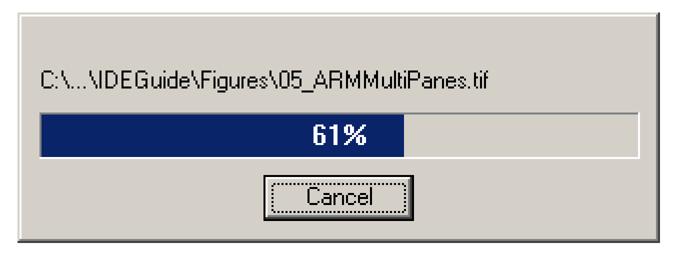


图 4-9　ads1.2 安装第 9 步

复制完成后在“ARM License Wizard”窗口单击“下一步”继续：

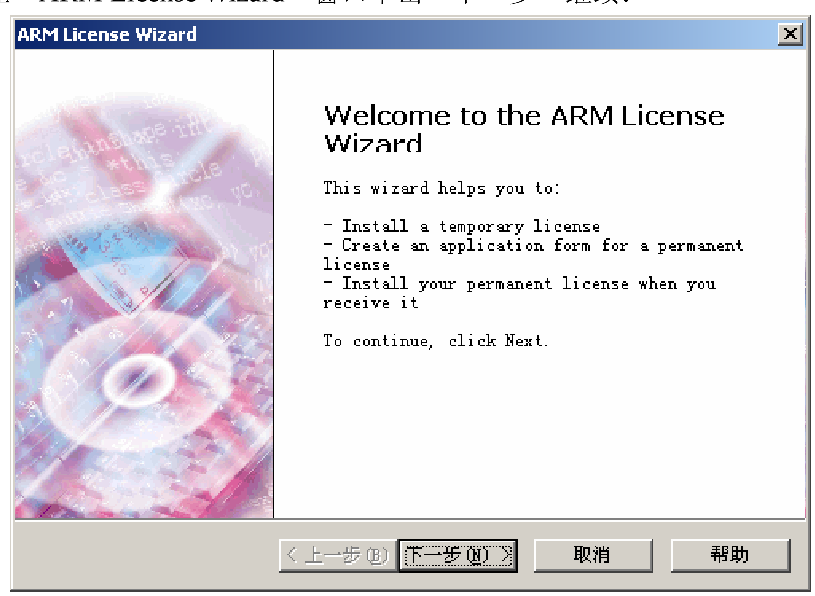


图 4-10　ads1.2 安装第 10 步

Choose Action 选择第一项“ Install License”，单击“下一步”继续：

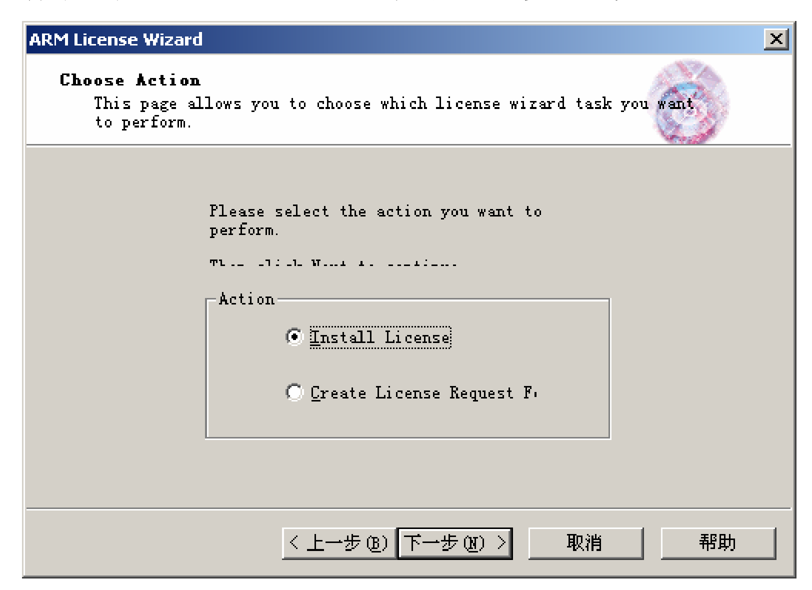


图 4-11　ads1.2 安装第 11 步

单击“Browse”打开 License fIle 即 LICENSE.DAT(在 ads12setup\CRACK 文件夹下)：

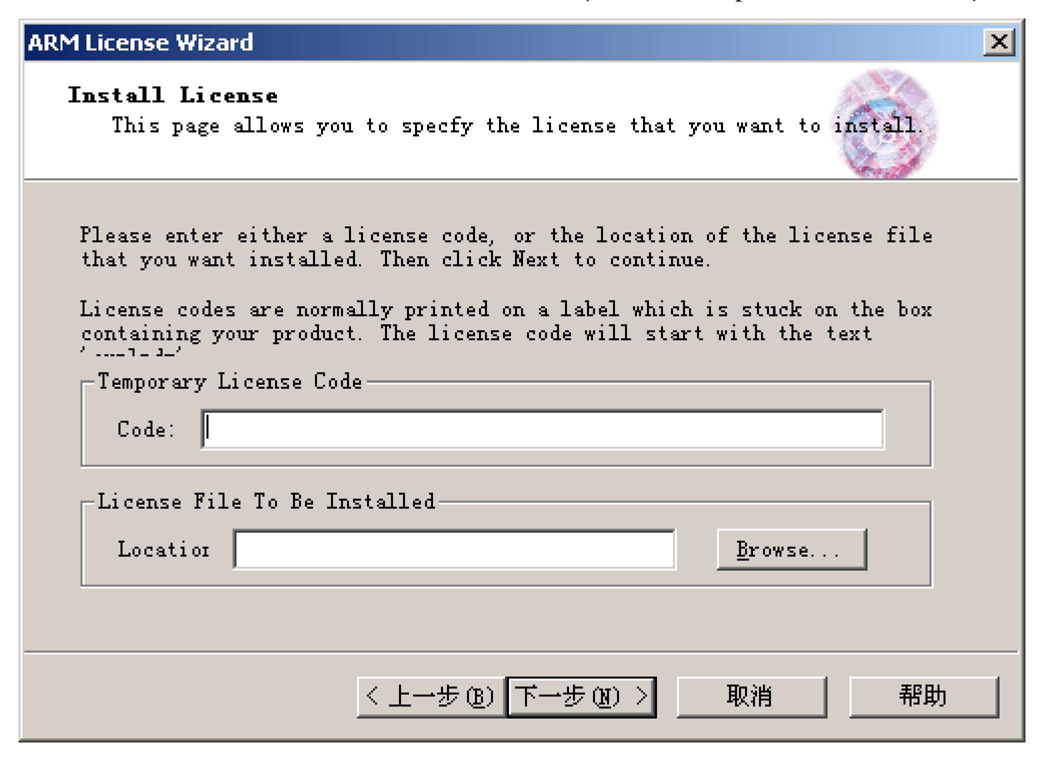


图 4-12 ads1.2 安装第 12 步

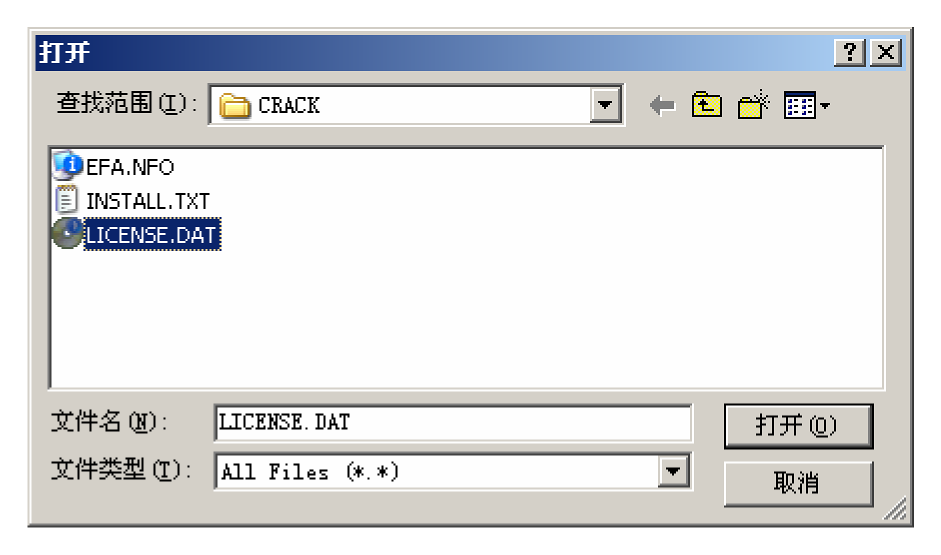


图 4-13 ads1.2 安装第 13 步

ARM License Wizard

Install License

This page allows you to specfy the license that you want to install.

Please enter either a license code, or the location of the license file that you want installed. Then click Next to continue.

License codes are normally printed on a label which is stuck on the box containing your product. The license code will start with the text

Temporary License Code

Code:

License File To Be Installed

Location C:\Documents and Settings\zhgy\桌面\ Browse...

< 上一步(B) 下一步(N) > 取消 帮助

图 4-14 ads1.2 安装第 14 步

单击“下一步”继续：

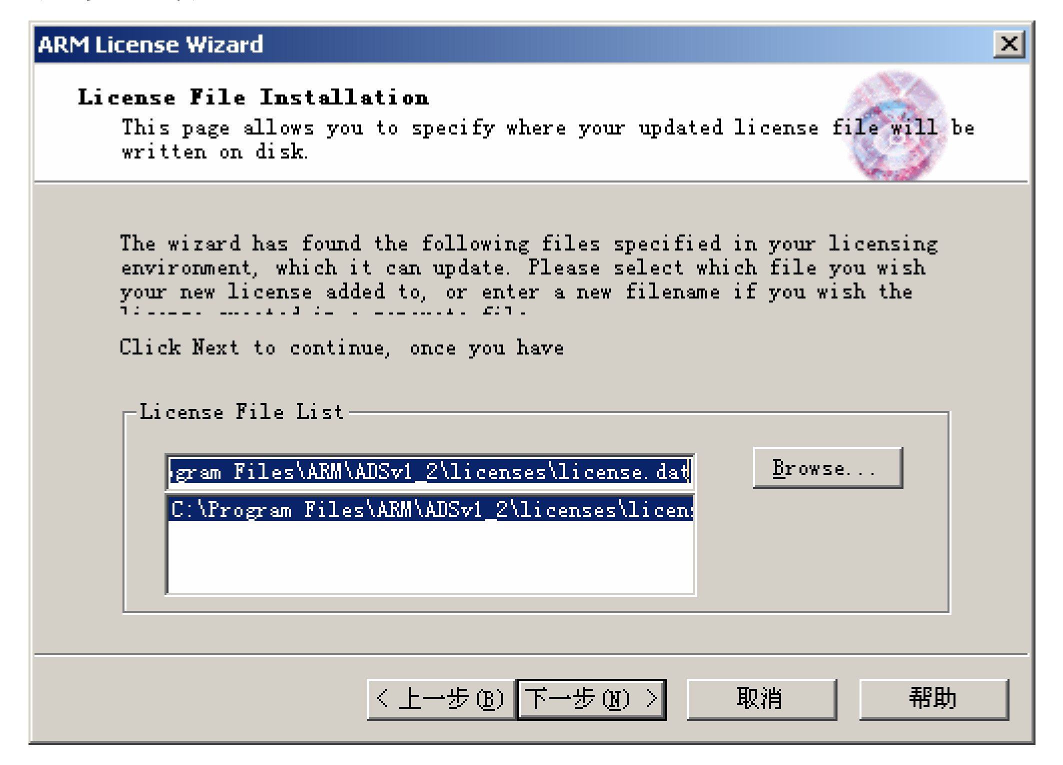


图 4-15 ads1.2 安装第 15 步

单击“完成”完成文件配置：

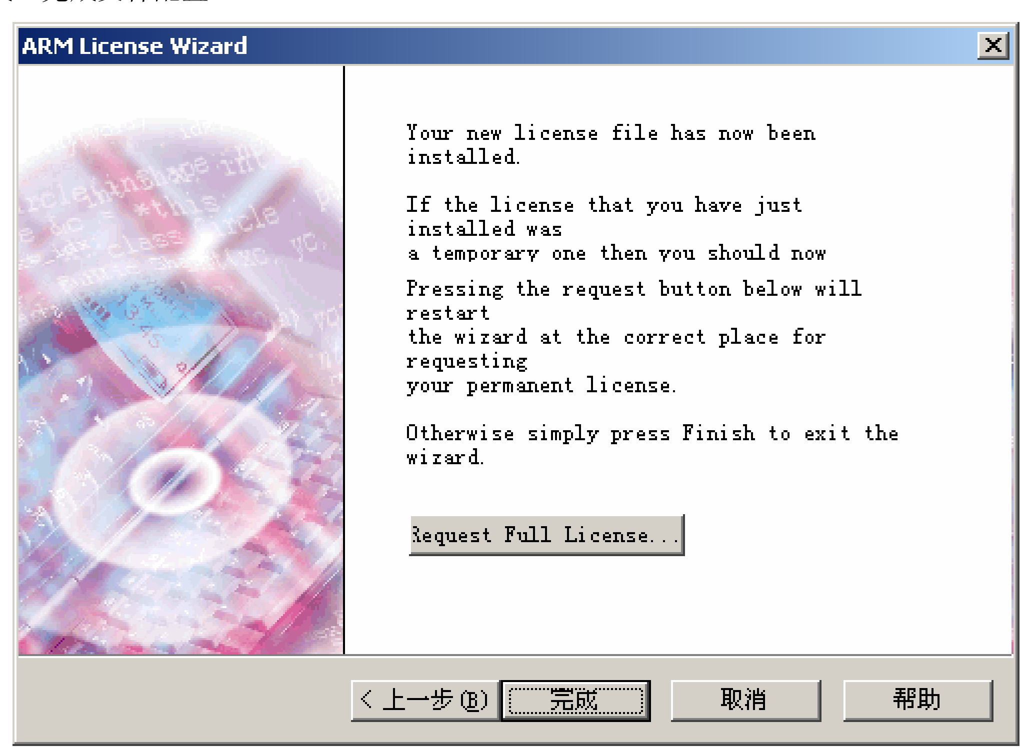


图 4-16 ads1.2 安装第 16 步

单击“Finish”完成全部安装过程：

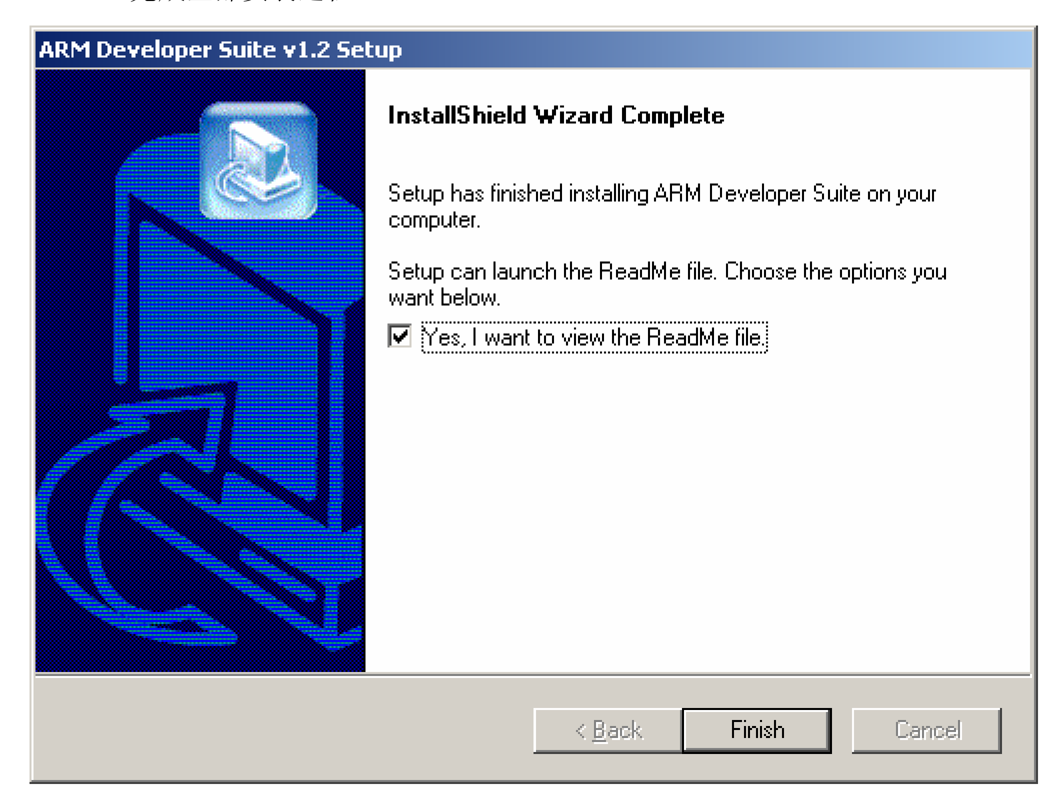


图 4-17 ads1.2 安装第 17 步

阅读 Readme 文件了解 ADS1.2 版本。



图 4-18 ads1.2 安装第 18 步

第 2 节 ADS 的设置及编译链接工程

以 LEDtest 工程为例，解压[脱离操作系统的试验代码\不同功能测试代码]下的 LEDtest.rar 然后双击打开其中的工程文件 LEDTEST.mcp：

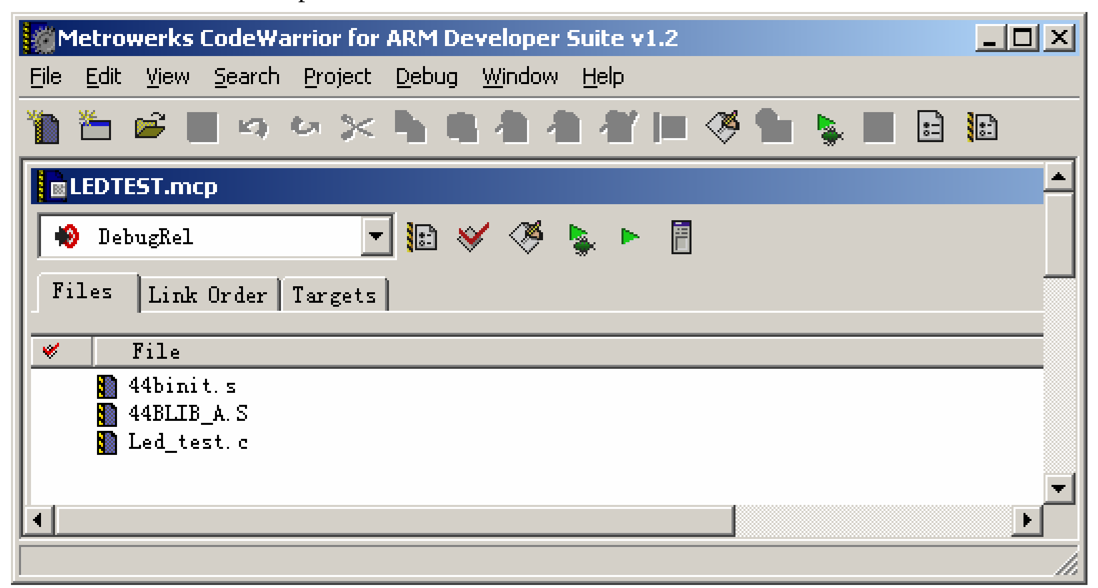


图 4-19 打开一个工程文件

设置

单击 Edit 菜单选择“DebugRel Settings…”(或单击工具栏中的按钮)进行设置：

注意：我们的工程文件都是设置好的，一般的情况你不用修改这些设置，在这里只是为了说明你建立一个工程需要设置的内容。

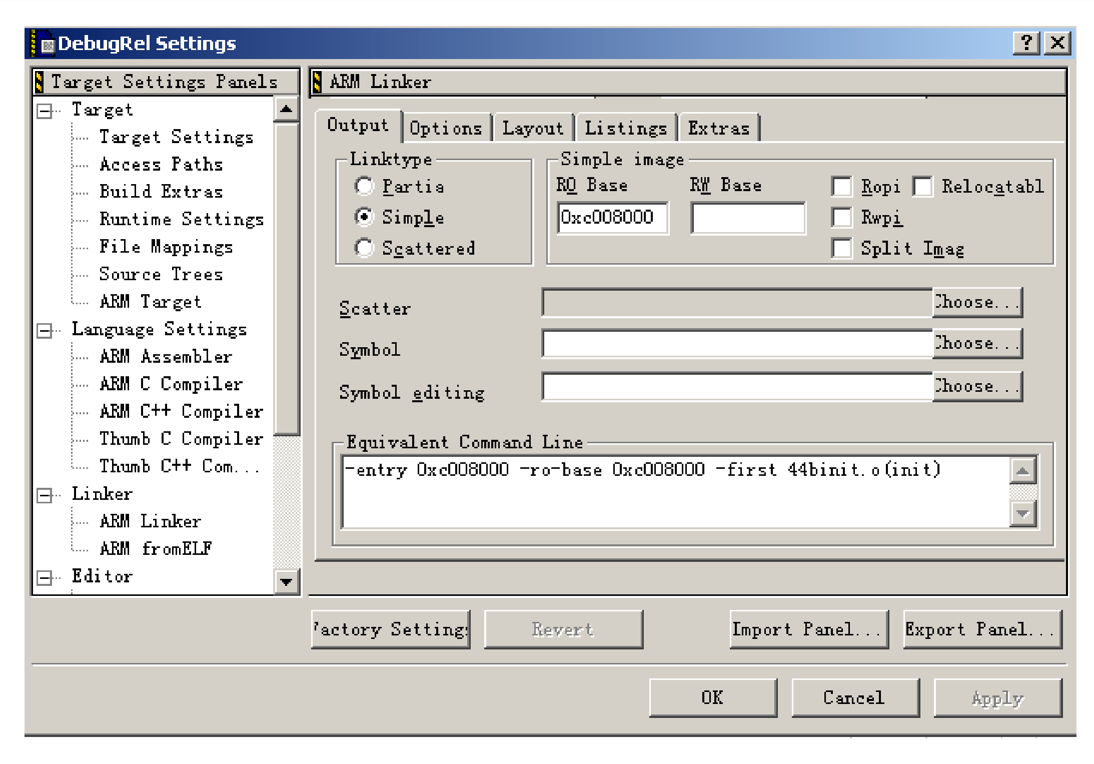


图 4-20 打开设置界面

在 Target→ARM Target→Output Name 选项的 Use project name 复选框前打勾，Output Type 选 Linker Output：

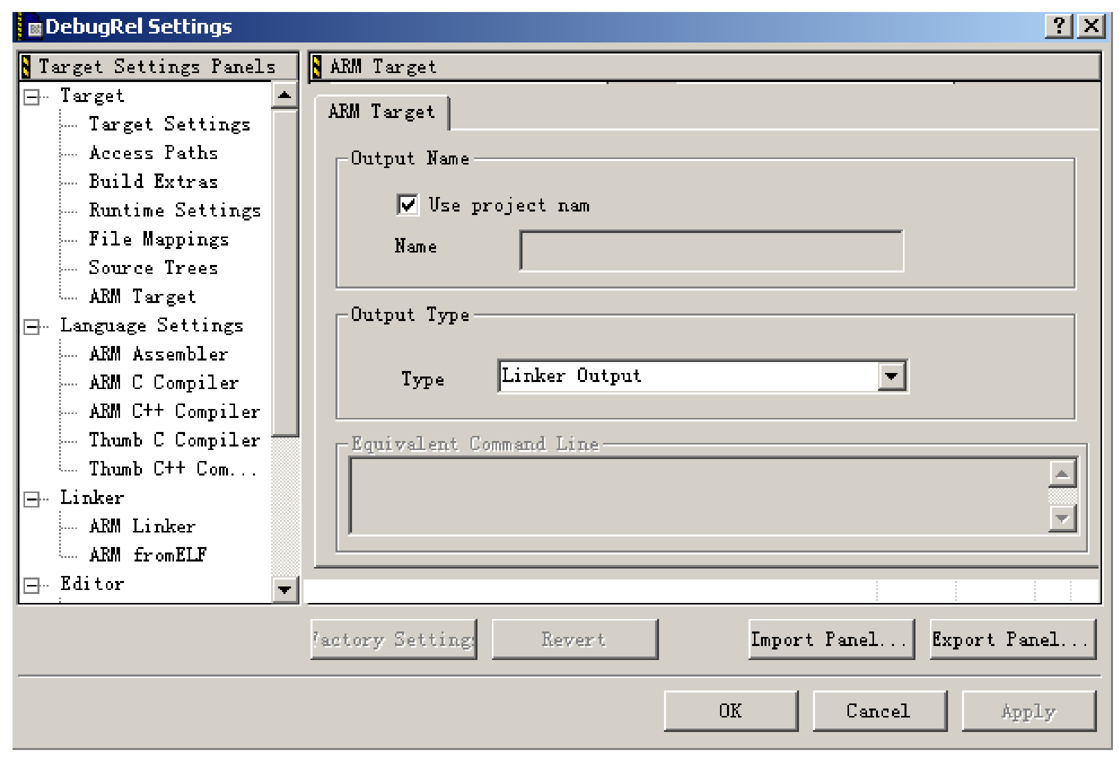


图 4-21 设置 ARM TARGET

LInker→ARM fromELF→Output format 选择 Plain binary：

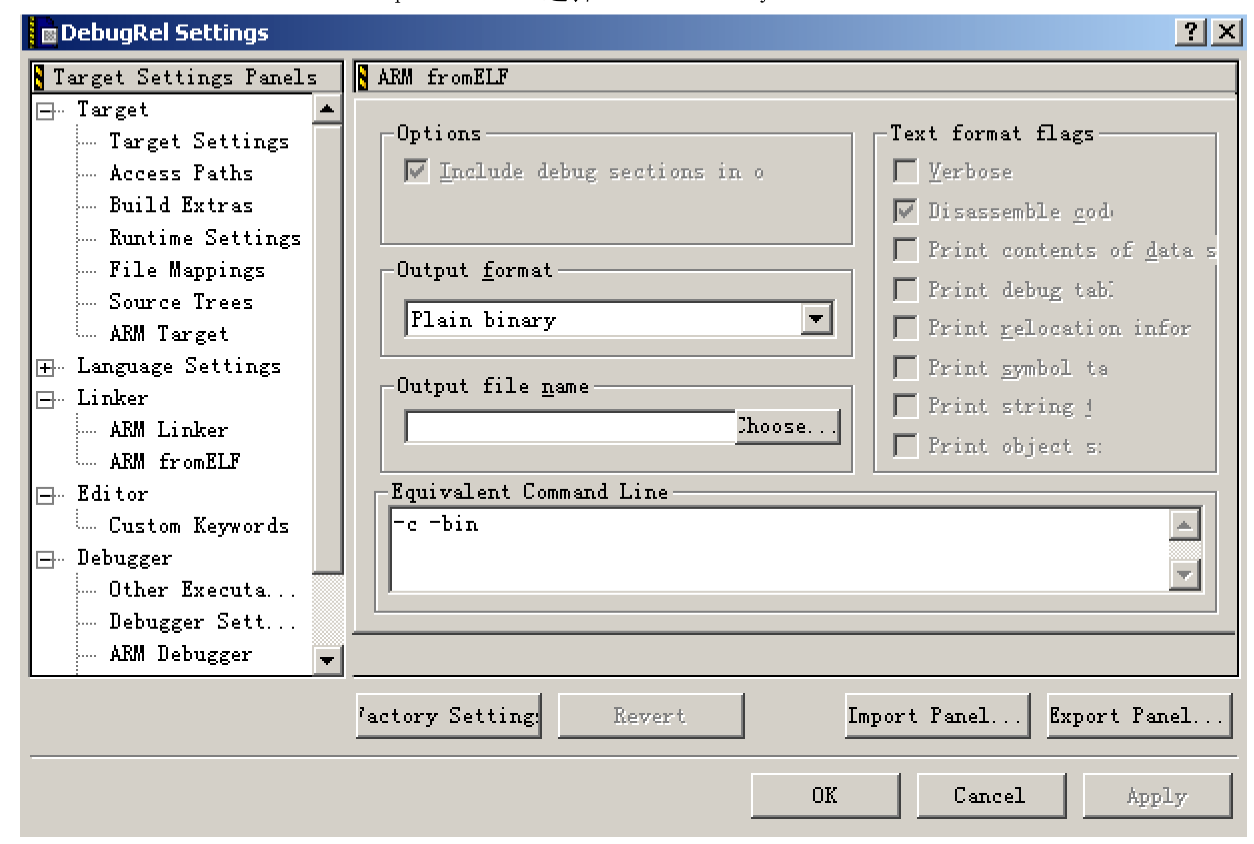


图 4-22 设置 ARM FormELF 选项

Target→Target Settings 的 Post-linker 选项选择 ARM fromELF，Linker 选项选择 ARM Linker：

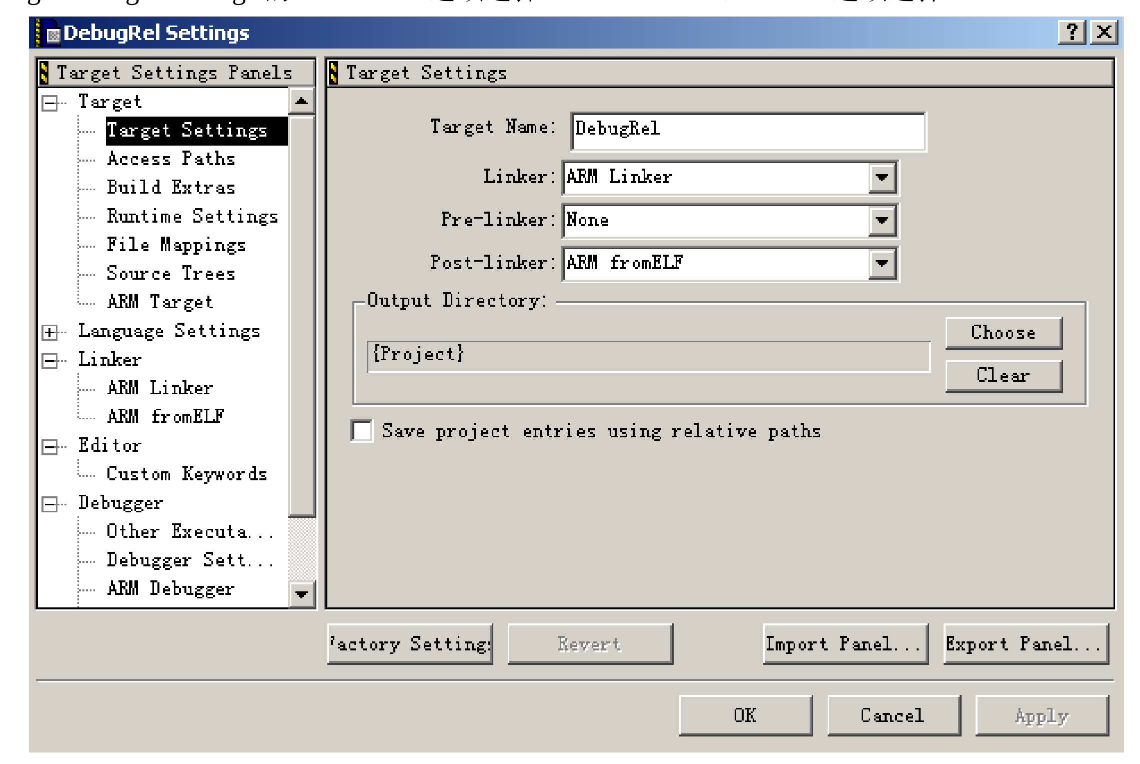


图 4-23 设置 target Setting 选项

Linker→ARM Linker→Output 页的 Linktype 选 simple，在 Simple image 的 RQ Base 文本框输入地址 0x0c008000：

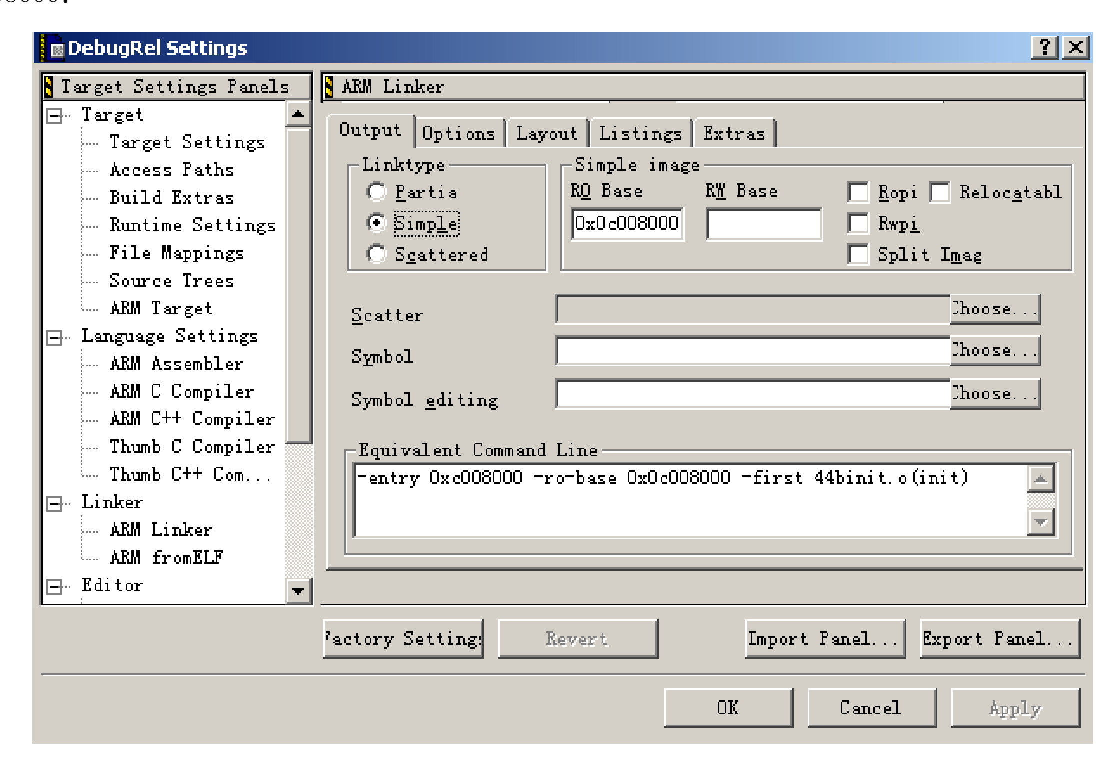


图 4-24 设置 ARM linker-Output

Linker→ARM Linker→Option 页的 Image entry point 文本框中输入 0x0c008000：

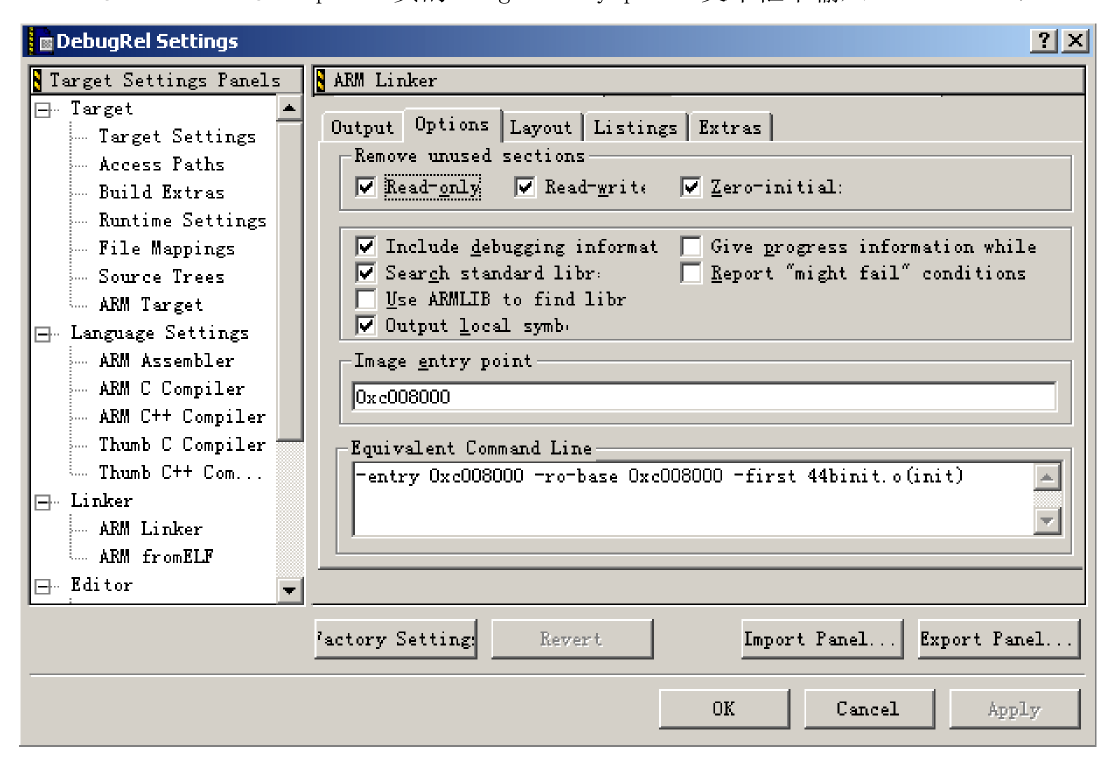


图 4-25 设置 ARM linker-options

Linker→ARM Linker→Layout 页的 Object/Symbol 文本框输入要执行的第一个文件的名称（本例是 44init.o）：

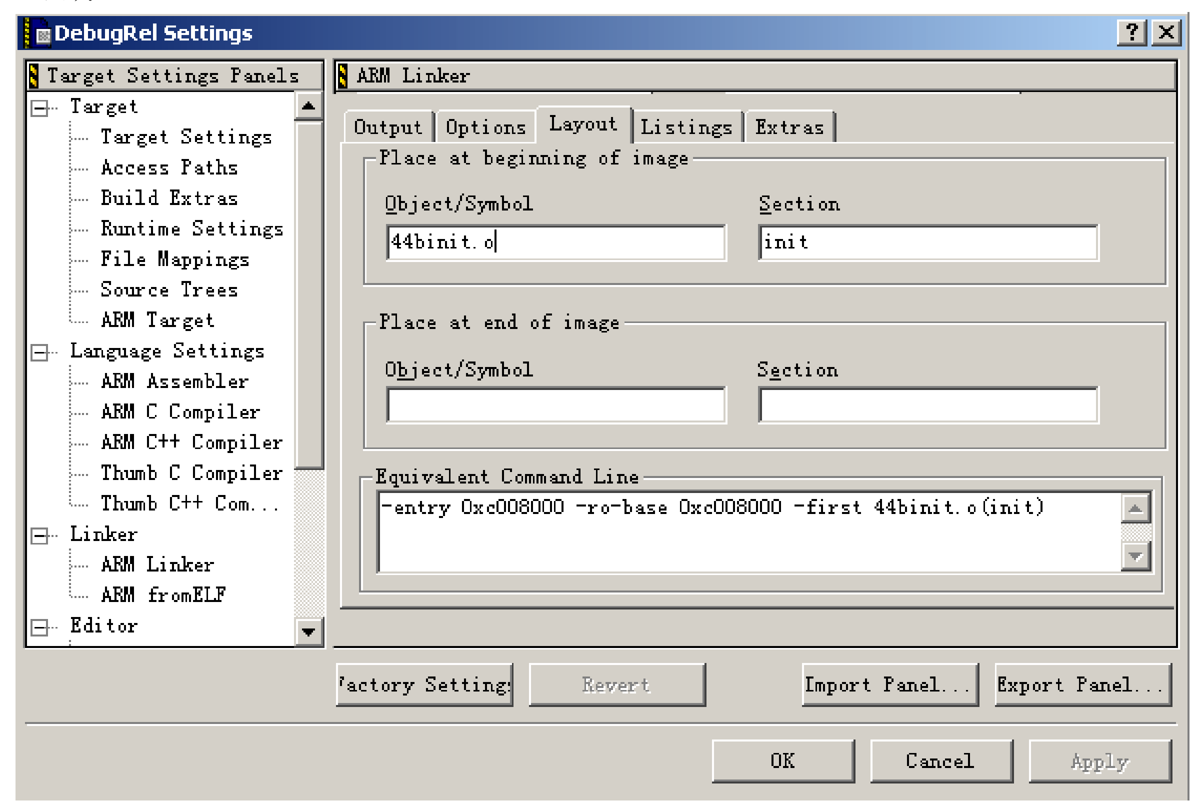


图 4-26 设置 ARM linker-options

完成设置后点击“OK”就可以进行编译链接了。

编译链接：

选中要编译链接的文件（即在文件前打勾），单击 Project 菜单选择 Make（或单击工具栏的 Make 按钮）编译链接工程：

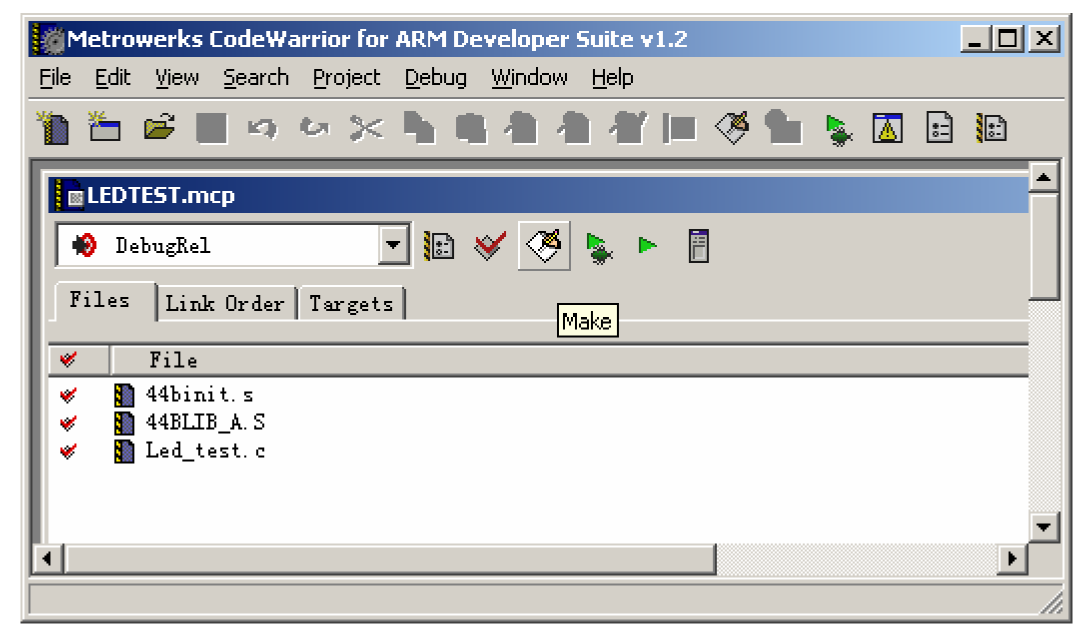


图 4-27 选择需要编译的文件

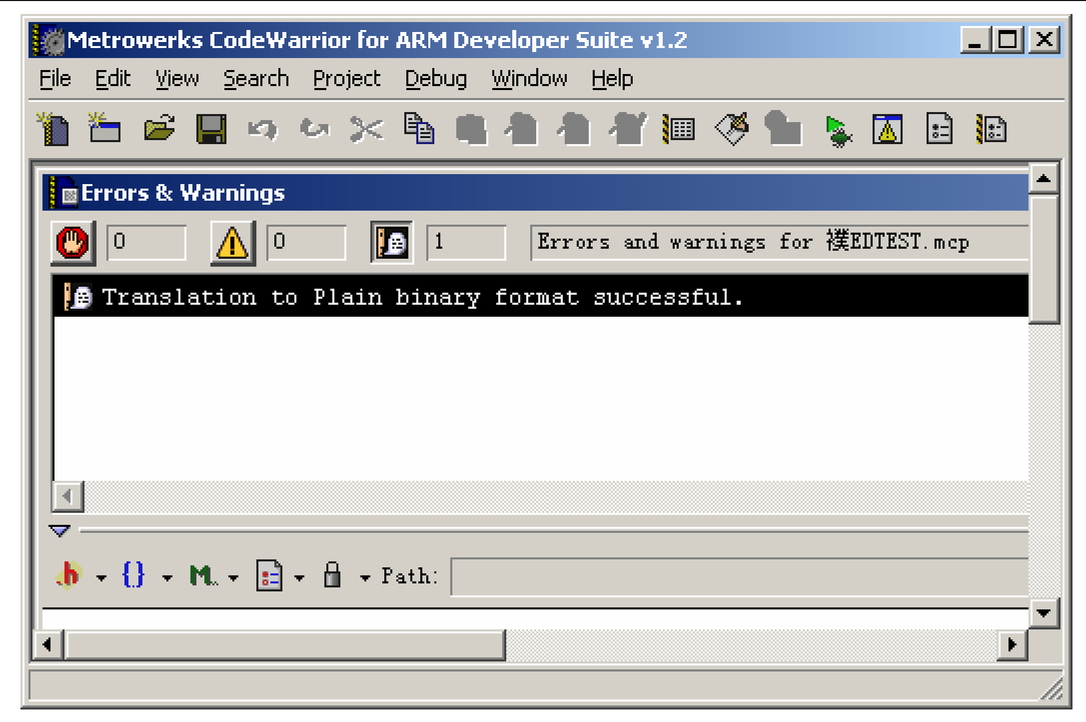


图 4-28 编译成功

在 disk3\LEDtest\LEDTEST_Data\DebugRel 文件夹下可以看到 Make 后生成的映像文件 LEDTEST.axf 和二进制文件 LEDTEST.bin，映像文件用于调试，二进制文件可以烧写到 S3C44B0 的 Flash 中运行。

第三节 ADS 的 AXD 设置及代码调试

首先通过并口线把 PC 与 Jtag 连接，然后将 Jtag 接到开发板的 Jtag 调试端口，接通开发板电源，打开电源开关。双击 ARMJtagDebugFinal (工具软件\JTAGARM.rar）文件夹下的 ARM7.exe，选择 Wiggler 模式，如果左边文本框显示“检测到：ARM7TDMI”便可以打开 AXD 了。



图 4-29 检测内核

单击 Option 菜单选择“Configer Target...”：

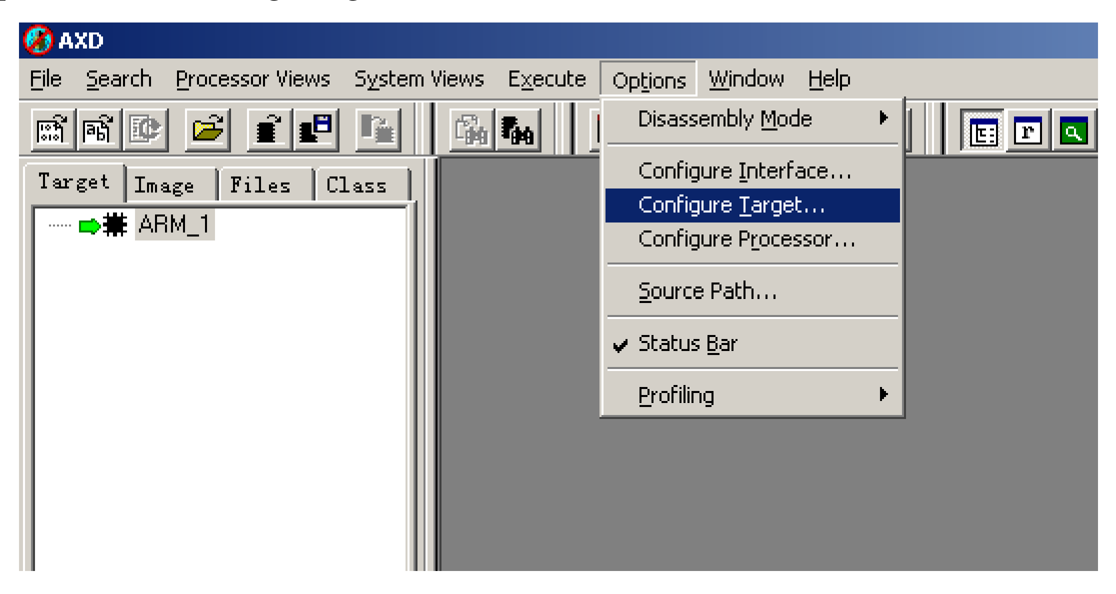


图 4-30 配置 AXD

Target 选择 ADP，然后单击 Configure 按钮进行配置：

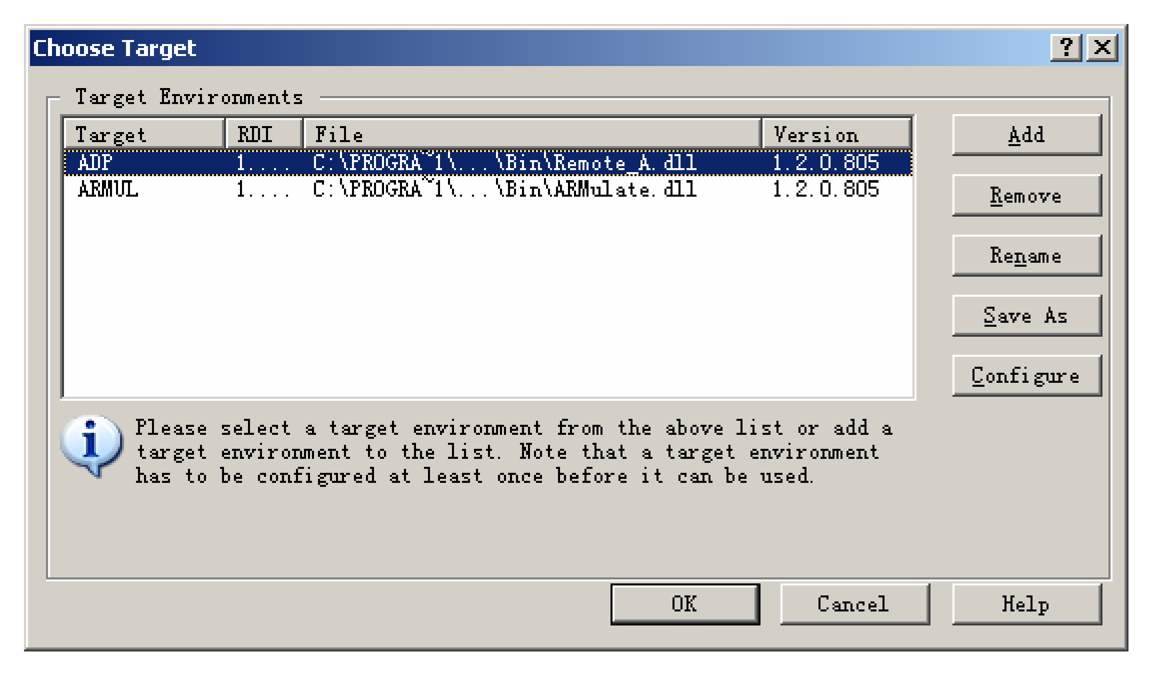


图 4-31 选择硬件仿真

单击“Select...”按钮选择驱动器：

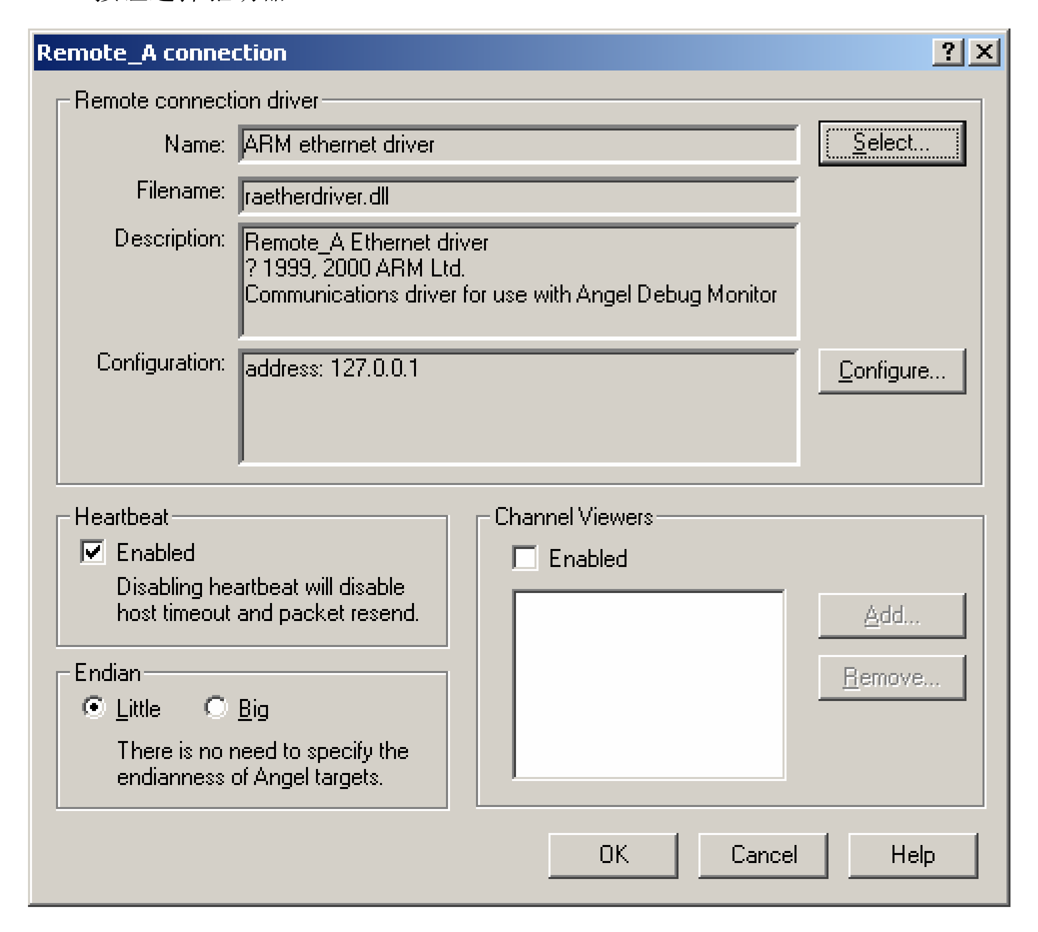


图 4-32 配置驱动程序

选择“ARM ethernet driver”后单击“OK”：

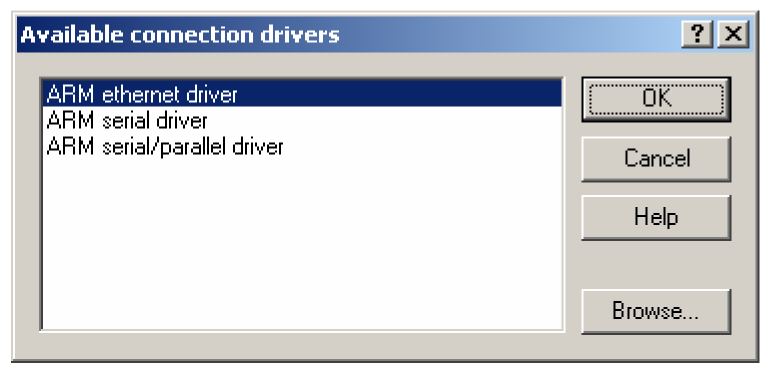


图 4-33 选择以太网驱动

在 Remote_A connection 窗口单击“Configure...”按钮设置 IP 地址，在 IP 地址文本框输入 127.0.0.1，单击“OK”完成配置。

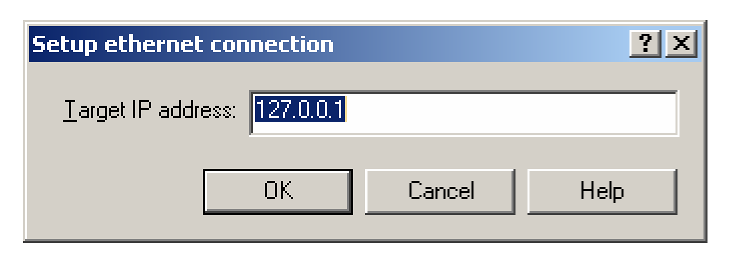


图 4-34 输入 ip:127.0.0.1 地址代表本机

单击 File 菜单选择“Load Image...”，打开 Load Image 对话框，找到要装载的.axf 映像文件（本例映像文件是 LEDTEST.axf（在 LEDtest\LEDTEST_Data\DebugRel\下面）），点击“打开”按钮就把映像文件装载到目标内存中了（注：ADS 不认中文目录）：

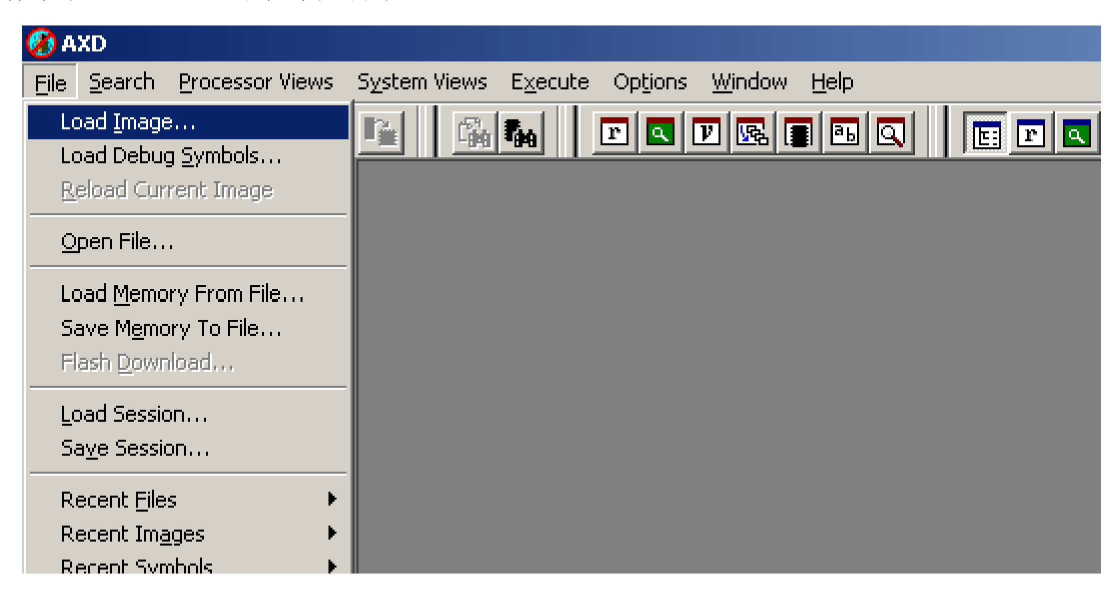


图 4-35 装载调试文件

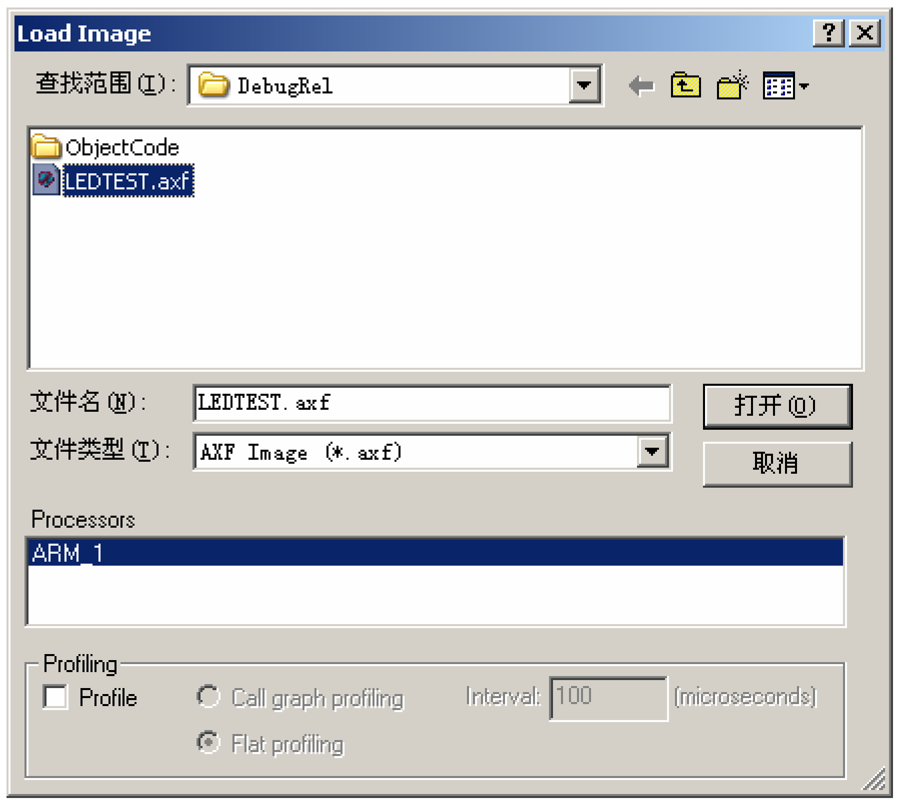


图 4-36 选择需要调式的文件

在所打开的映像文件中有一个蓝色的箭头指示当前执行的位置：

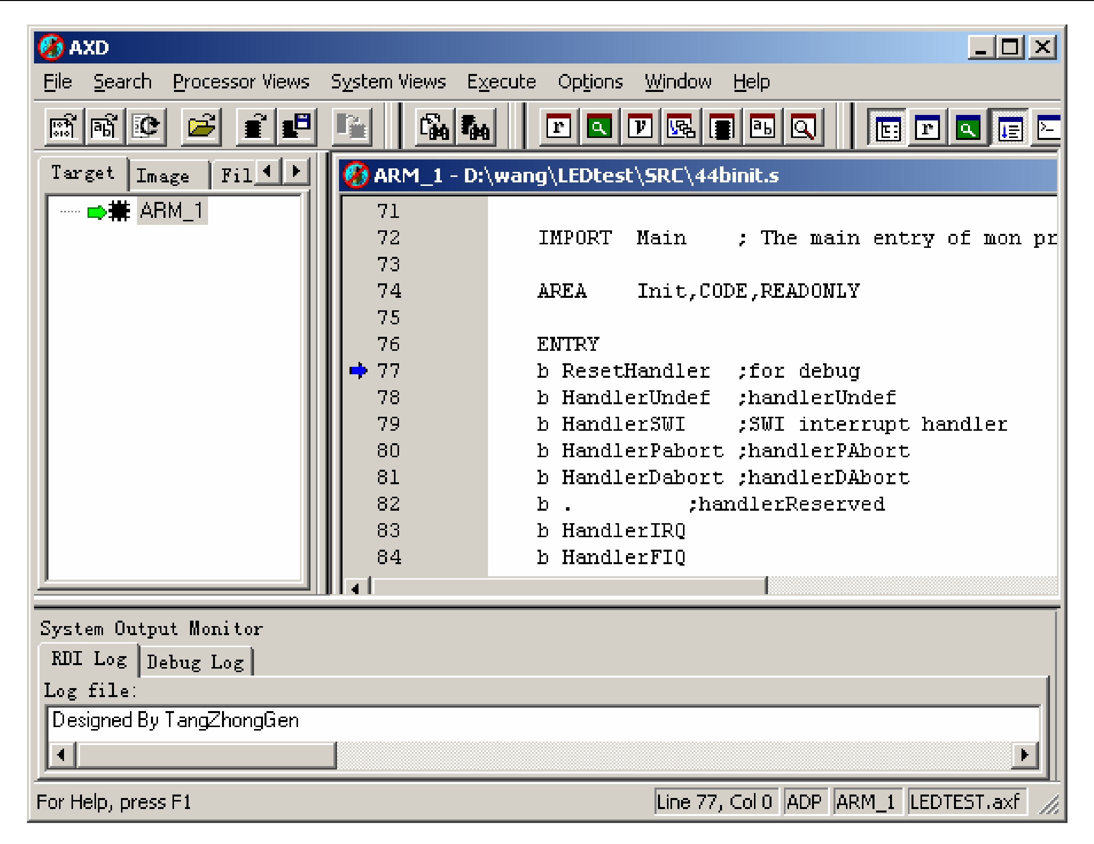


图 4-37 装载文件成功

在 Excute 菜单中选择“Step In”或单击工具栏中的“Step In”按钮对程序进行单步调试（进入函数内部）：

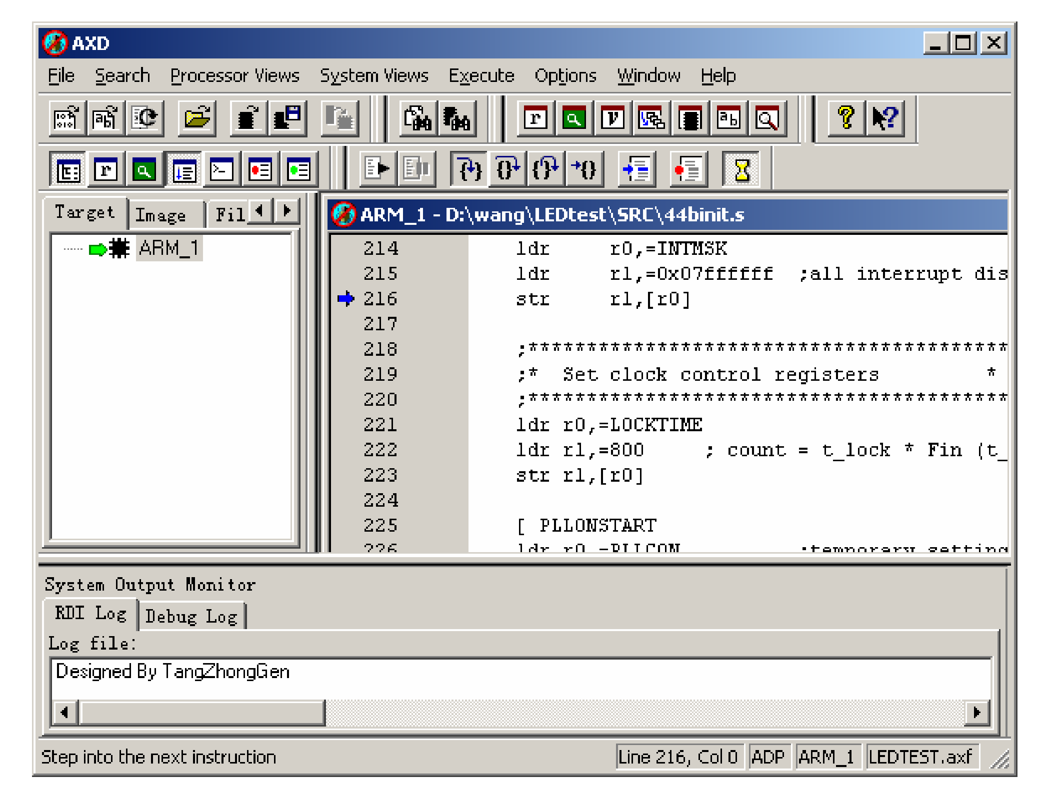
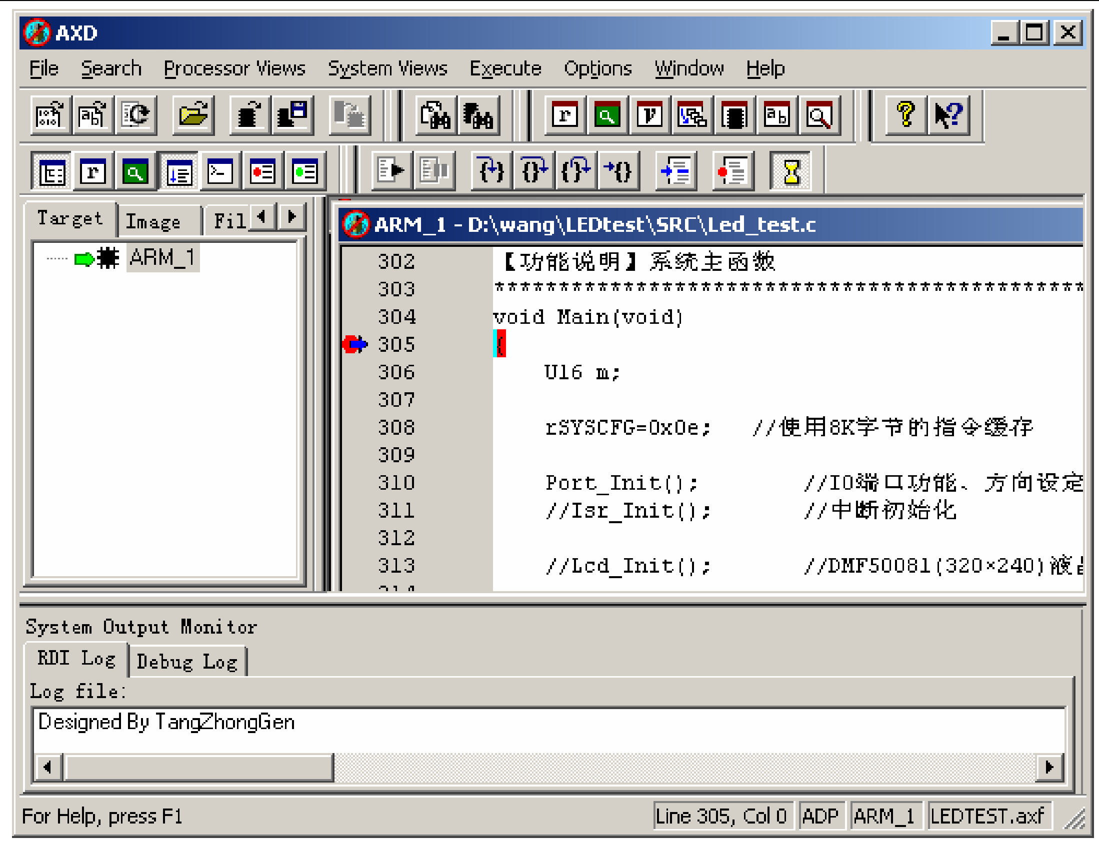


图 4-38 用 AXD 调试系统

单击“Go”按钮全速运行代码（主函数入口处系统自动设置断点）：



在要进行断点设置的代码处双击，该位置就会出现一个红色的实心圆点表明该处为断点(在断处双击即取消断点) ；单击“Go”代码运行到断点处暂停：

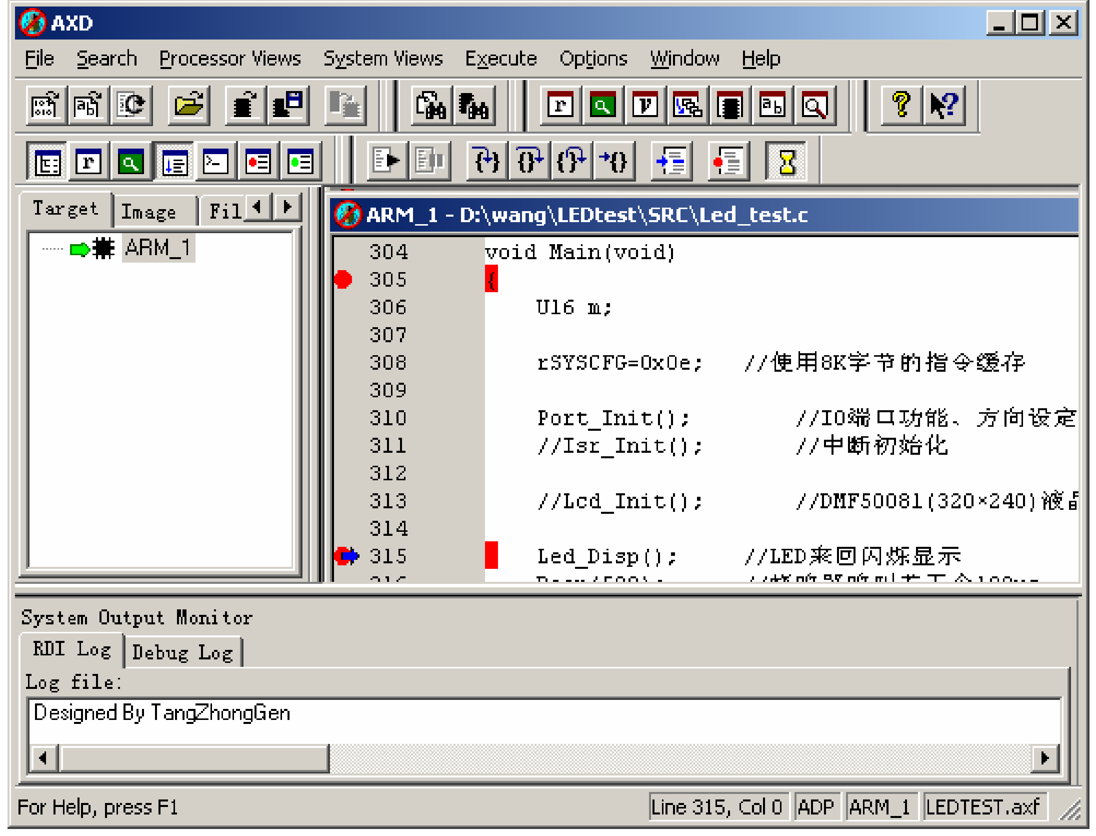


图 4-38 用 AXD 调试系统

单击“Go”代码继续向下运行：

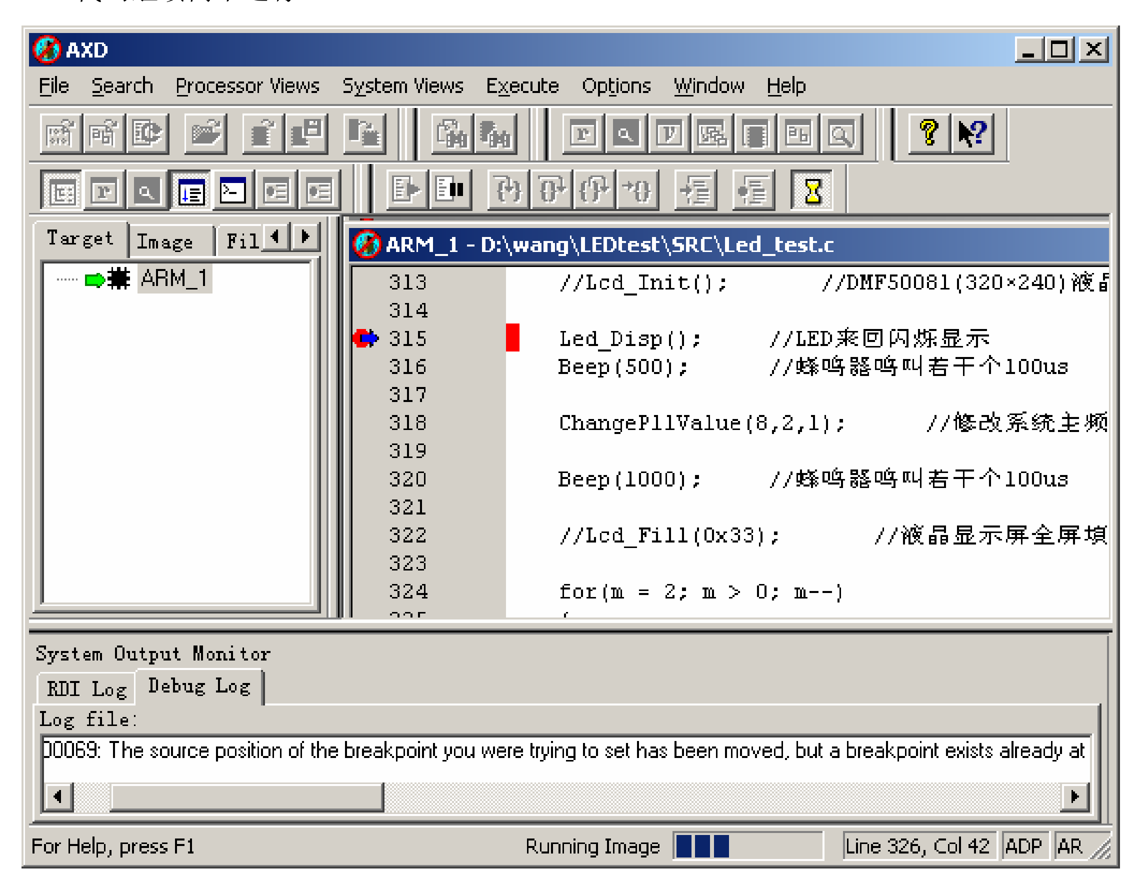


图 4-39 正在全速运行

单击“Stop”结束运行。

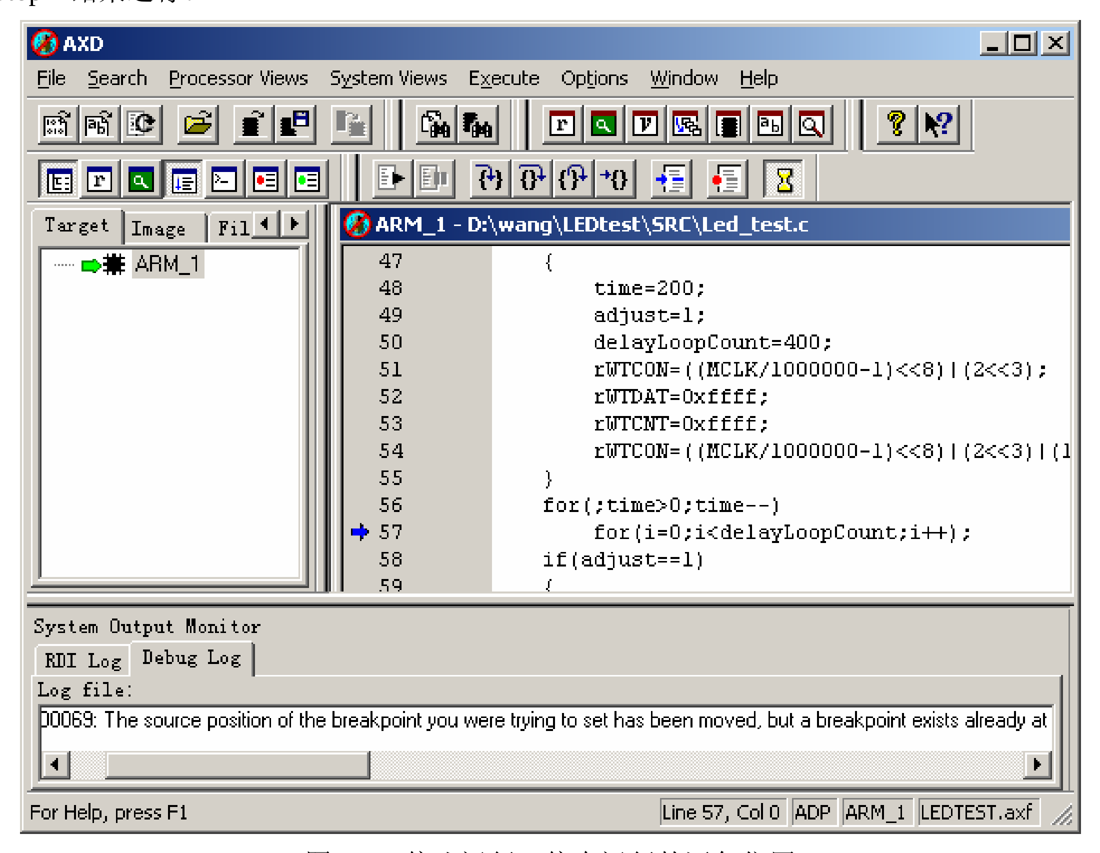


图 4-40 停止运行，停在运行的语句位置

第五章功能试验程序详细讲解

第一节 LED 程序演示

主程序 led_test.c 代码如下（原代码位于：[脱离操作系统的试验代码\不同功能测试代码]）：

```
#include "inc\44b.h"    //主程序必须包含这两个头文件
#include "inc\option.h"
static int delayLoopCount = 400;
void Delay(int time)    //这是一个通用的延时函数
{
	int i,adjust=0;
	if(time==0)
	{
		time=200;
		adjust=1;
		delayLoopCount=400;
		rWTCON=((MCLK/1000000-1)<<8)|(2<<3);
		rWTDAT=0xffff;
		rWTCNT=0xffff;
		rWTCON=((MCLK/1000000-1)<<8)|(2<<3)|(1<<5);
	}
	for(;time>0;time--)
		for(i=0;i<delayLoopCount;i++);
	if(adjust==1)
	{
		rWTCON=((MCLK/1000000-1)<<8)|(2<<3);
		i=0xffff-rWTCNT;
		delayLoopCount=8000000/(i*64);
	}
}
```

/***
由 S3C44B0 的原理图知 PC 口各引脚的功能，再对应芯片手册(S3C44B0.pdf)中对 PCONC（PC 口控制寄存器）每一位的描述（P225）来设置 PCONC 的值，所以 rPCONC =1111 1111 1111 0101 1111 1111 0101 0101B=0xfff5ff55。

PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8
nCTS0	nRTS0	RXD1	TXD1	nCTS1	nRTS1	nEL_ON	nDISP_ON
11	11	11	11	11	11	01	01

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
VD4	VD5	VD6	VD7	LED2	LED1	LED0	D12SUSPD
11	11	11	11	01	01	01	01

***/

```
void Port_Init(void)    //该函数设定 I/O 端口（PC 口）的功能
{
    rPDATC = 0x0000; //将 PC 口数据寄存器的 PC15~PC0 全设为 0
    rPCONC = 0xfff5ff55;
}
```

/***
三个 LED 点亮/熄灭状态设置：由原理图可知 PC1、PC2、PC3 三个口依次对应与 LED0、LED1、LED2 连接，且当 PC 口为高电平时对应的 LED 点亮，当 PC 口为低电平时对应的 LED 熄灭。所以 rPDATC=rPDATC&0xfffD 时，PC1＝0，LED0 熄灭；rPDATC=rPDATC|0x02 时，PC1＝1，LED0 点亮。
***/

```
void Led_Display(int LedStatus)
{
    if((LedStatus&0x01)==0x01)      //PC1 状态设置
        rPDATC=rPDATC&0xfffD;           // LED0 熄灭
    else
        rPDATC=rPDATC|0x02;             // LED0 点亮

    if((LedStatus&0x02)==0x02)      //PC2 状态设置
        rPDATC=rPDATC&0xfffB;
    else
        rPDATC=rPDATC|0x04;

    if((LedStatus&0x04)==0x04)      //PC3 状态设置
        rPDATC=rPDATC&0xfff7;
    else
        rPDATC=rPDATC|0x08;
}
```

/***
LED 来回闪烁显示：调用 Led_Display(　)函数时，将其参数值传递给 LedStatus 来选择对 PC1~PC3 哪个口进行状态设置，进而控制对应的 LED 点亮 / 熄灭。
***/

```
void Led_Disp(void)
{
    Led_Display(0x04);          //LED 点亮/熄灭状态设置
    Delay(3000);        //延时若干个 100us
    Led_Display(0x02);
    Delay(3000);
    Led_Display(0x01);
    Delay(3000);
}

void Main(void)        //系统主函数
{
    Port_Init();       //I/O 端口功能、方向设定

    while(1)

    {
    Led_Disp();       //LED 来回闪烁显示
    }
}
```

在文件编辑 (Text File）窗口编辑好代码后，在工程窗口选中要编译链接的文件（即在文件前打勾），单击 Project 菜单选择 Make（或单击工具栏的 Make 按钮）编译链接工程，如显示“Translation to Plain binary format successful.”(即 0 Erros 、0 Warnings)编译链接成功，否则根据提示回编辑窗口修改代码：

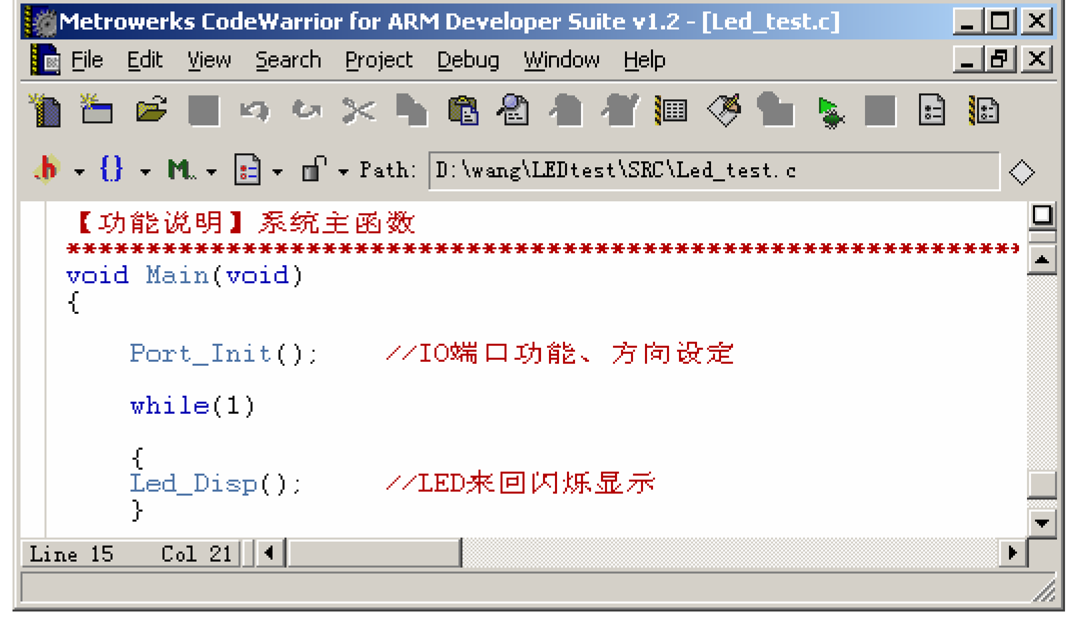


图 5-1-1 显示主程序

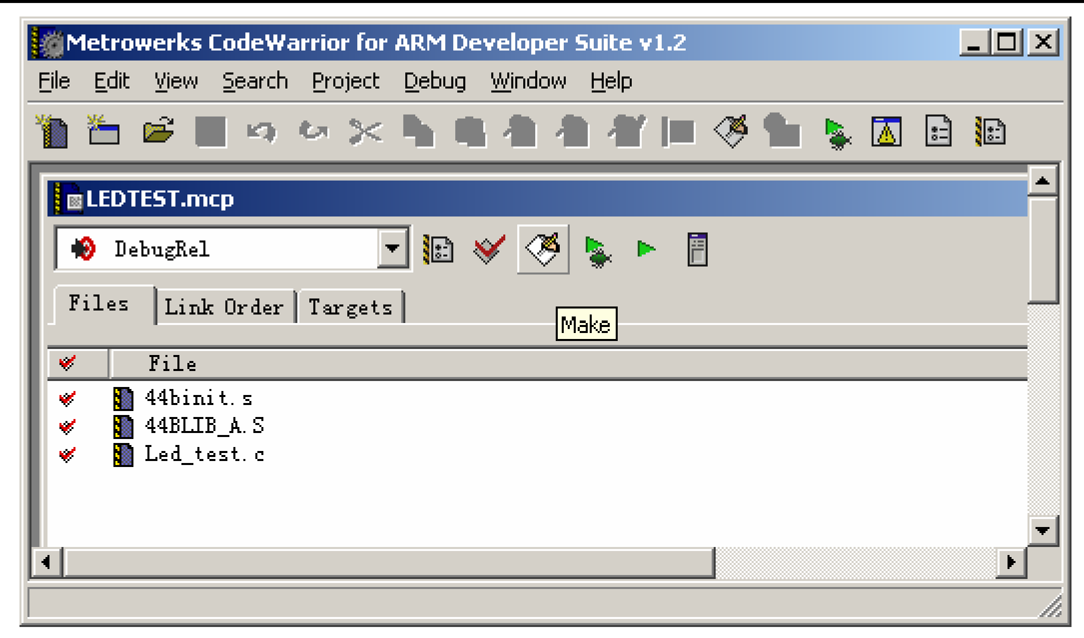


图 5-1-2 选择需要编译的文件

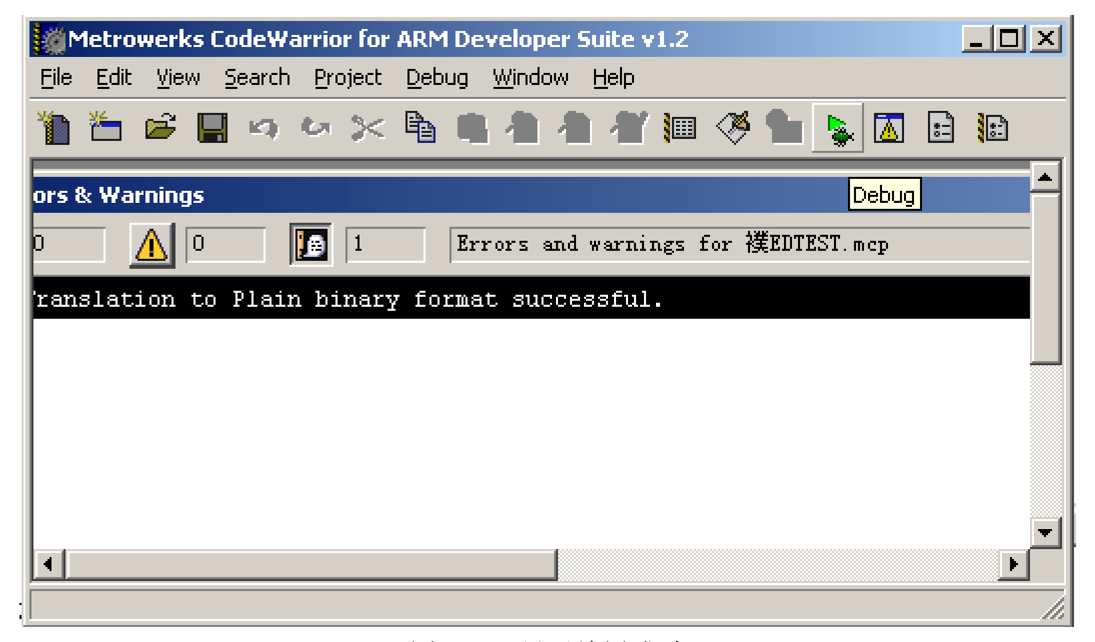


图 5-1-3 显示编译成功

接下来单击 Project 菜单选择 Debug（或单击工具栏的 Debug 按钮）用 AXD 对代码进行调试。注意：在调试之前请确认开发板通过 Jtag、并口线与 PC 正确连接，ARM7 调试代理 v1.5 检测到：ARM7TDMI，且 AXD 正确配置。

文件被加载后程序自动跳转到蓝色箭头所指位置也就是当前执行的位置（这是由 ARM Linker 中设定的入口地址所决定的）：

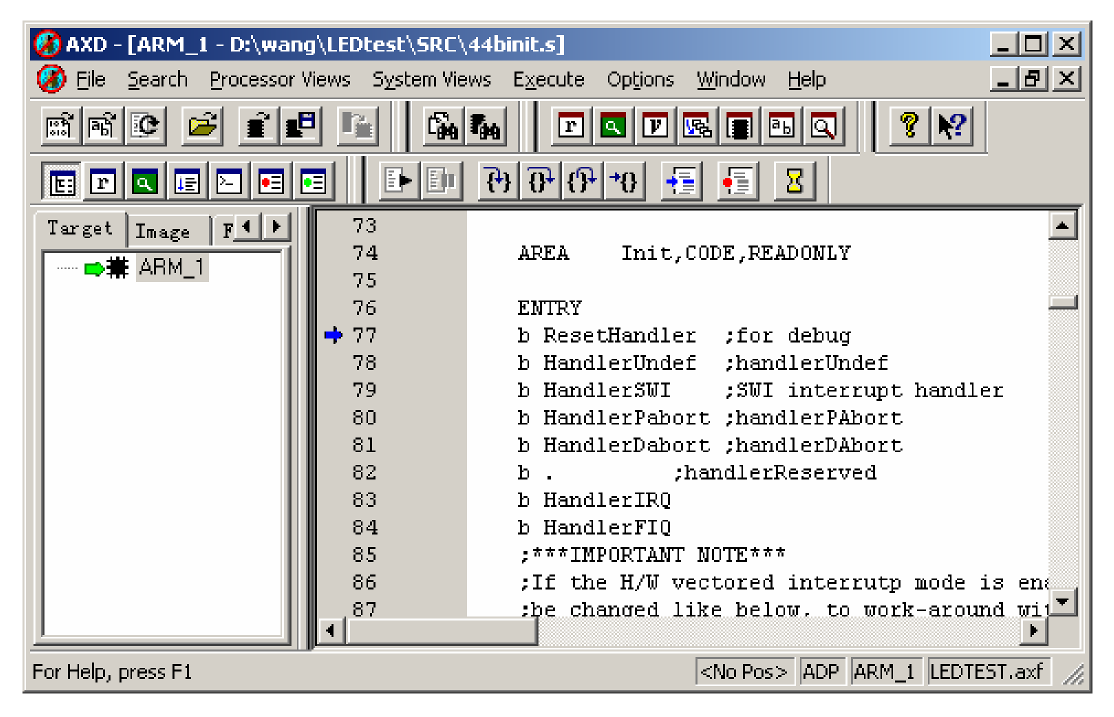


图 5-1-4 AXD 加载成功

单击”Step In”按钮单步运行代码，跳转到第 210 句：

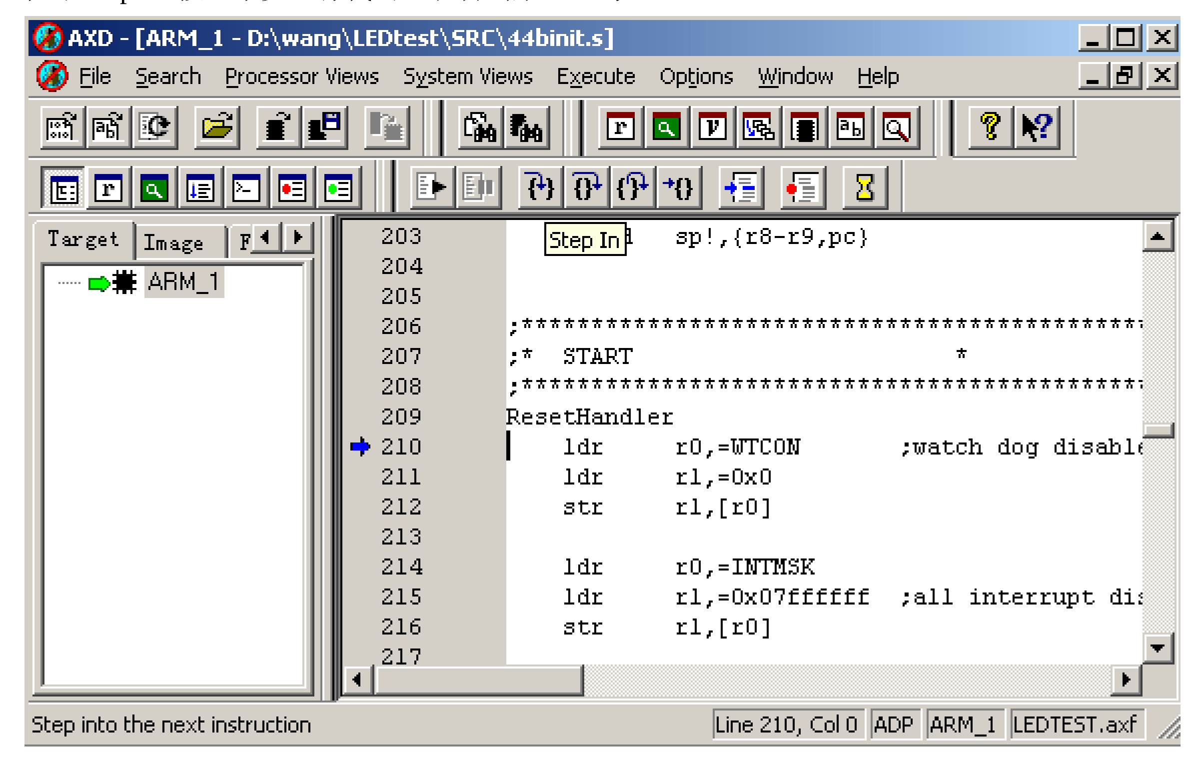


图 5-1-5 AXD 执行程序

运行到第 280 句时根据指令跳转到主函数：

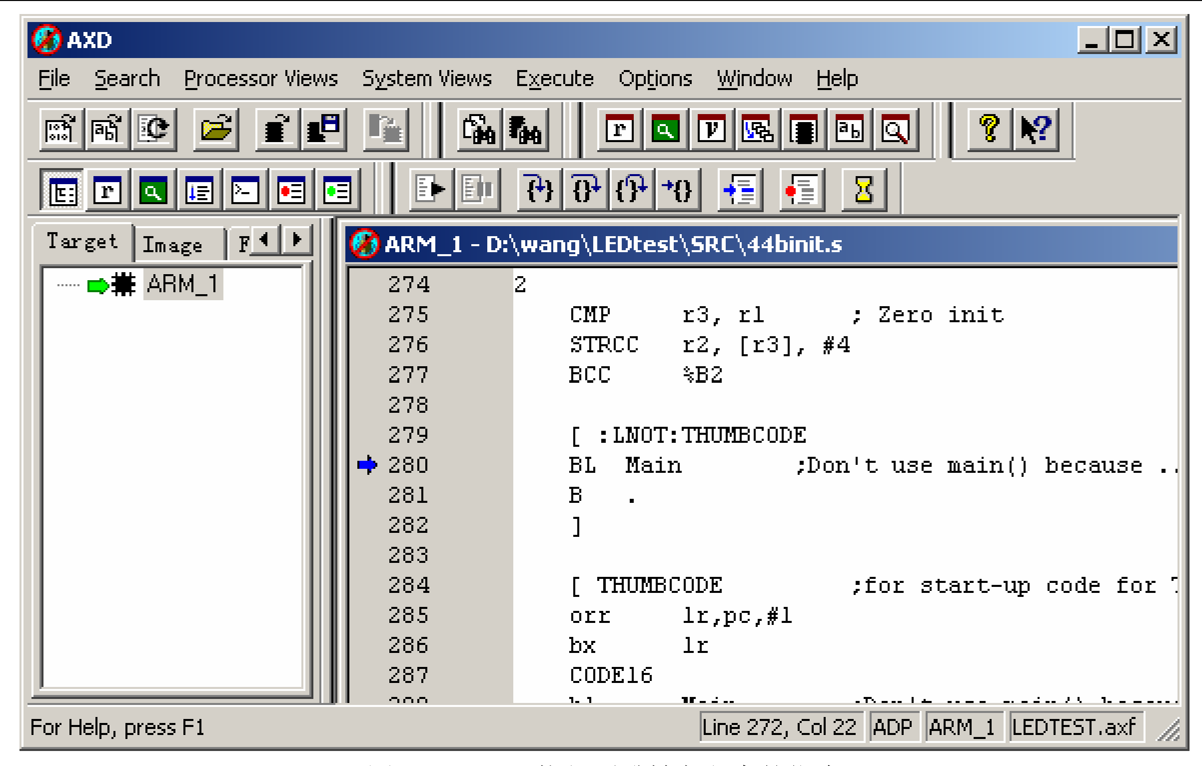


图 5-1-6 AXD 执行到跳转主程序的指令

主函数入口处系统自动设置断点，单击"Step In"（或按 F8）调用 Port_Init()函数：

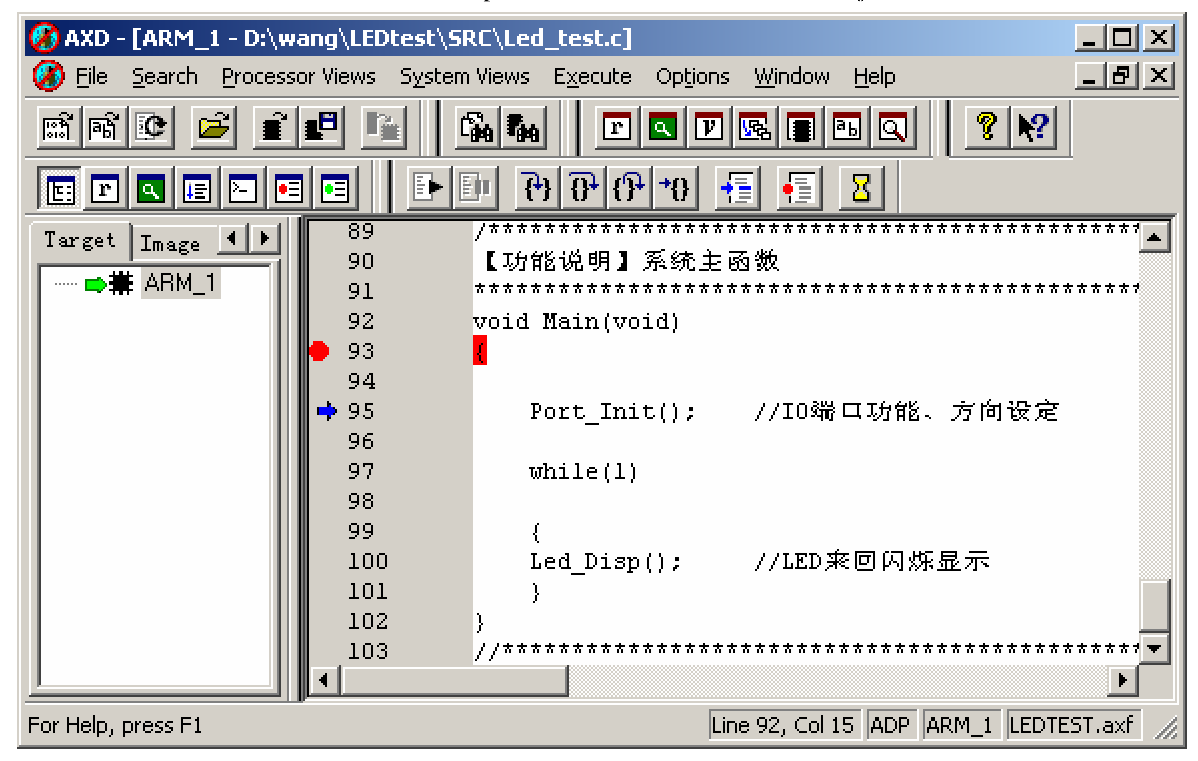


图 5-1-7 开始执行主函数

单击 Processor Views 菜单选择 Memory，在 Memory Start address 选择框中输入 0x1d20010（查看说明书可知这是 rPCONC 寄存器的起始地址）后回车，可以查看 PC 口寄存器 rPDATC 和 rPCONC 的内容及其变化，读数据的时候注意高地址中存放的是高字节，低地址中存放的是低字节。从下图可看出执行完第 47 句后 rPCONC 寄存器的值由 0f05ff55 变为 fff5ff55。

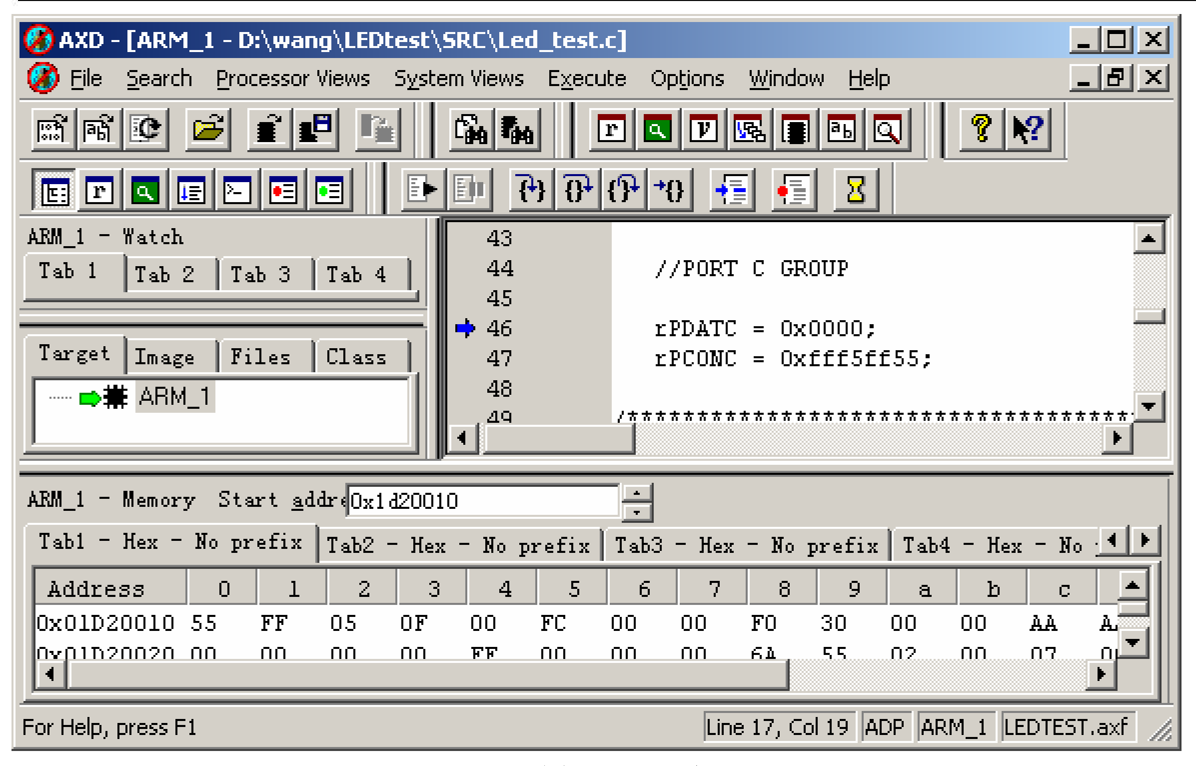


图 5-1-8 调试 LED

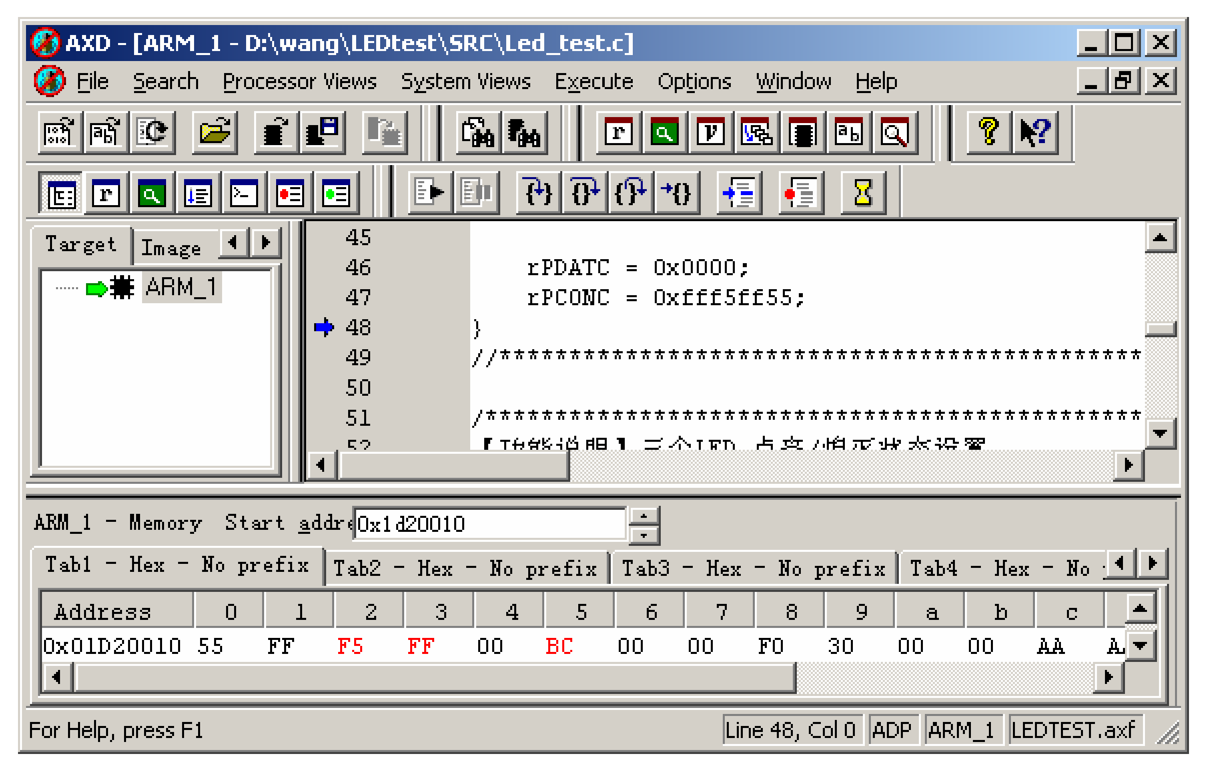


图 5-1-9 调试 LED

按 F8 单步运行代码，程序跳转到主程序调用 Led_Disp()函数：

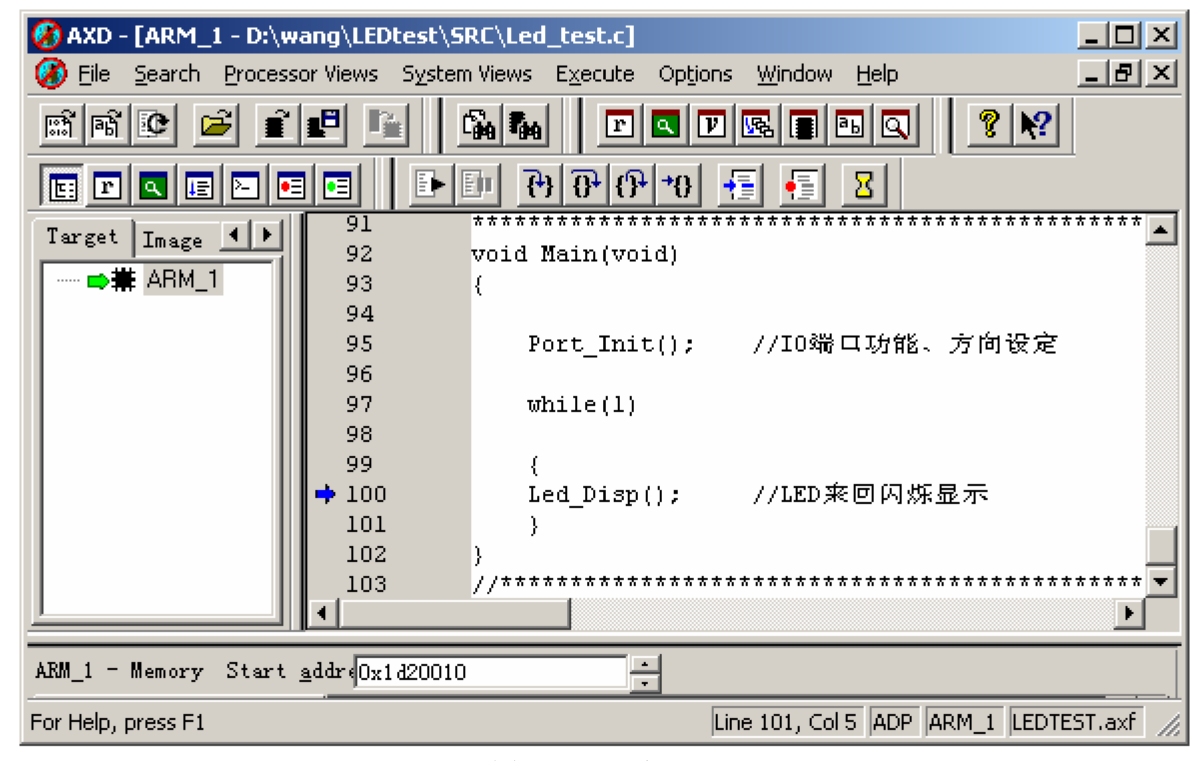


图 5-1-9 调试 LED

单击"step in"进入到 Led_Disp()函数内部，调用 Led_Display()函数：

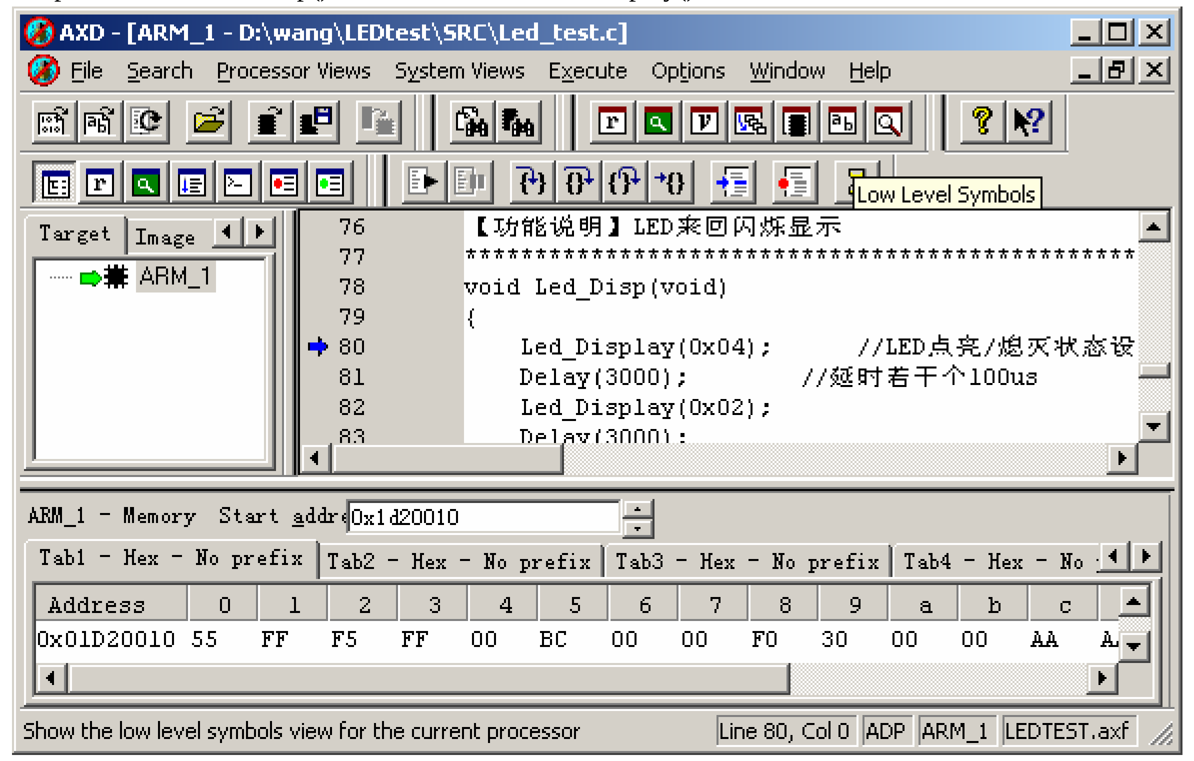


图 5-1-10 调试 LED

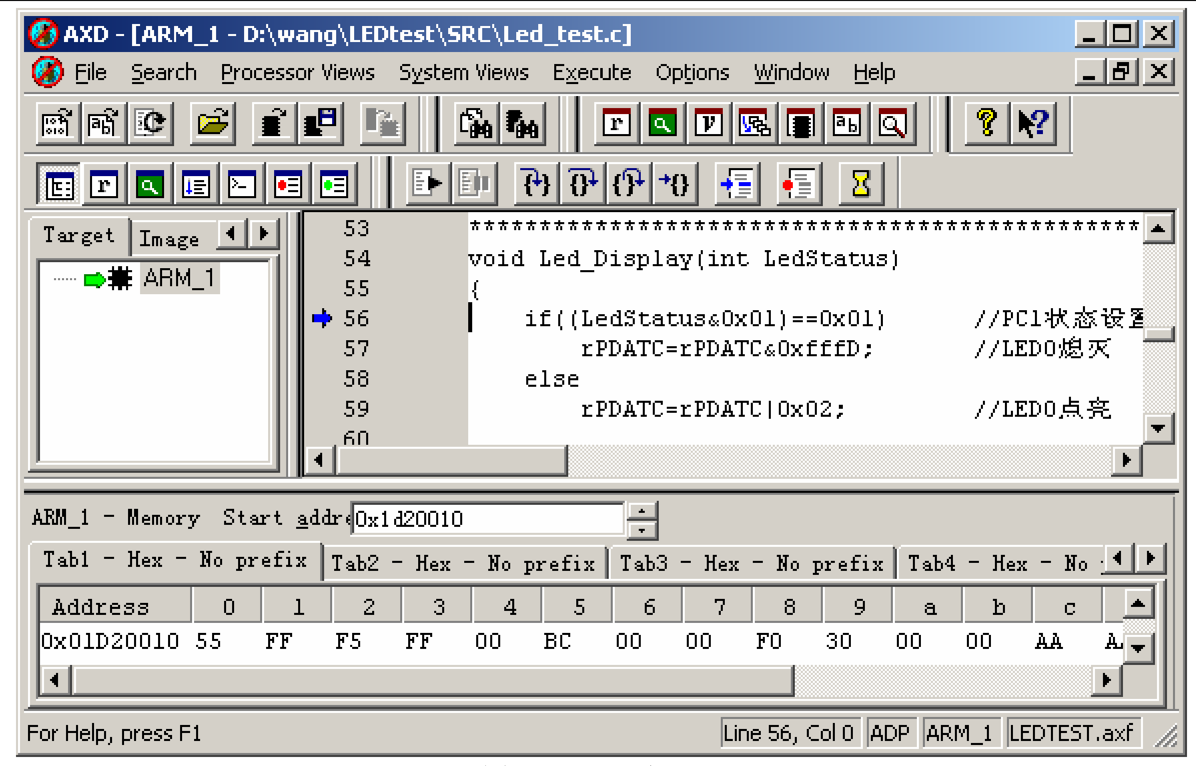


图 5-1-11 调试 LED

因为此时 Ledstatus 的值为 0x04，所以程序执行第 59 句，执行完第 59 句后第一个灯(LED0)点亮，rPDATC 寄存器的低八位值由 00 变为 02。要查看变量 Ledstatus 的值可以用鼠标选中它，然后点击鼠标右键选择“add to watch”这样就可以在右侧的 watch 窗口中看到它的值随着程序的运行在变化（查看本例中变量 i 和 time 的值的变化情况也用此方法）。

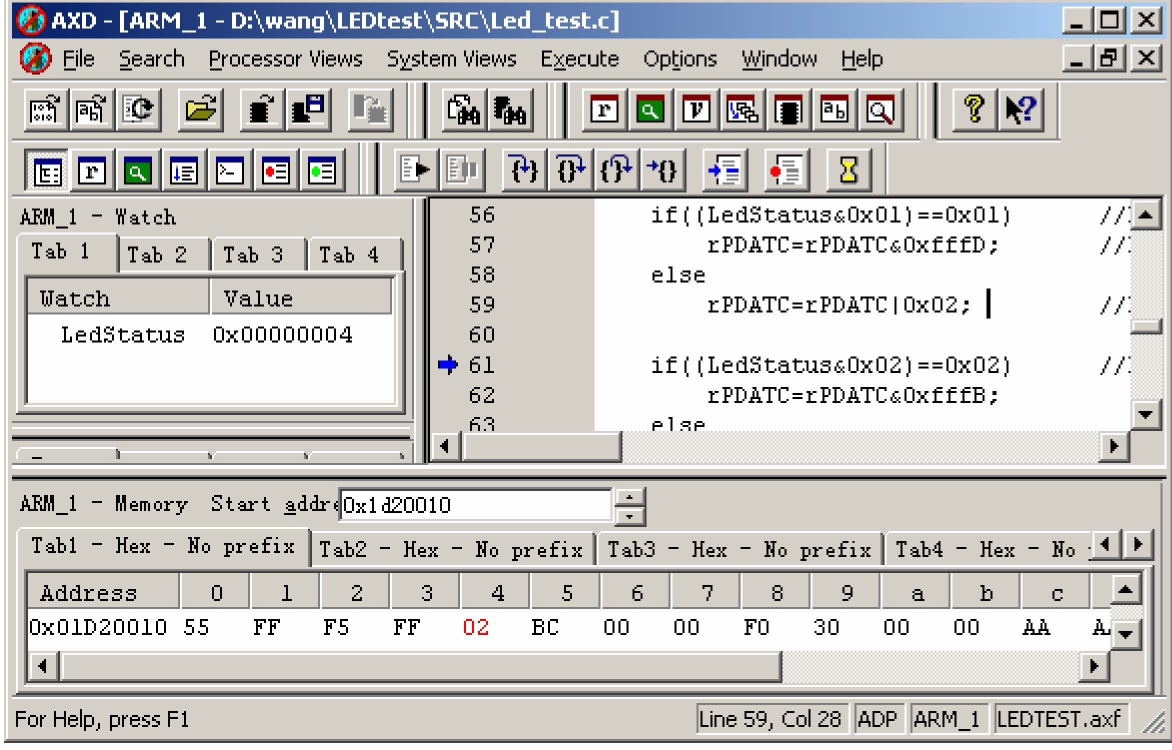


图 5-1-12 调试 LED

执行完第 64 句后第二个灯(LED1)点亮， rPDATC 寄存器的低八位值由 02 变为 06：

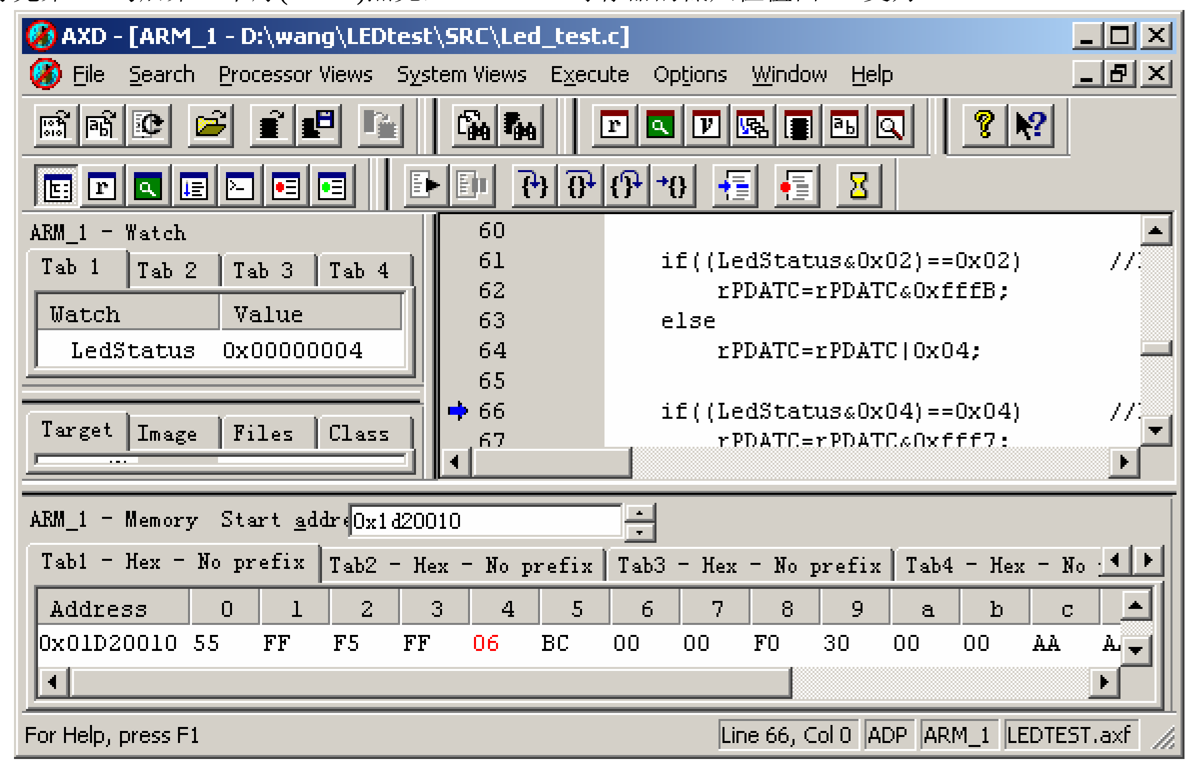


图 5-1-13 调试 LED

Ledstatus 的值为 0x04，所以程序执行第 67 句第三个灯(LED 2)为熄灭状态：

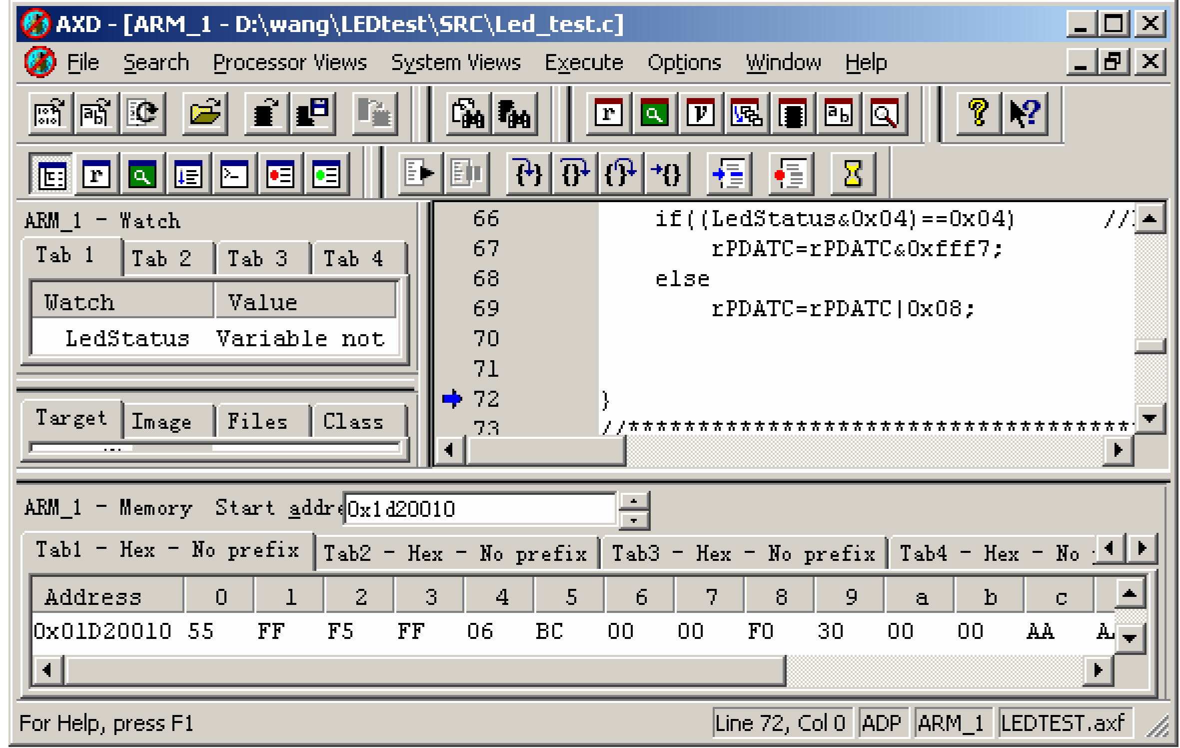


图 5-1-14 调试 LED

运行完 Led_Display()函数后单击”Step In”跳回到 Led_Disp()函数，然后单击”step”执行完 Delay()函数（此时点击”step”单步执行 Led_Display()函数，但不进入调用的函数 Delay()函数的内部）：

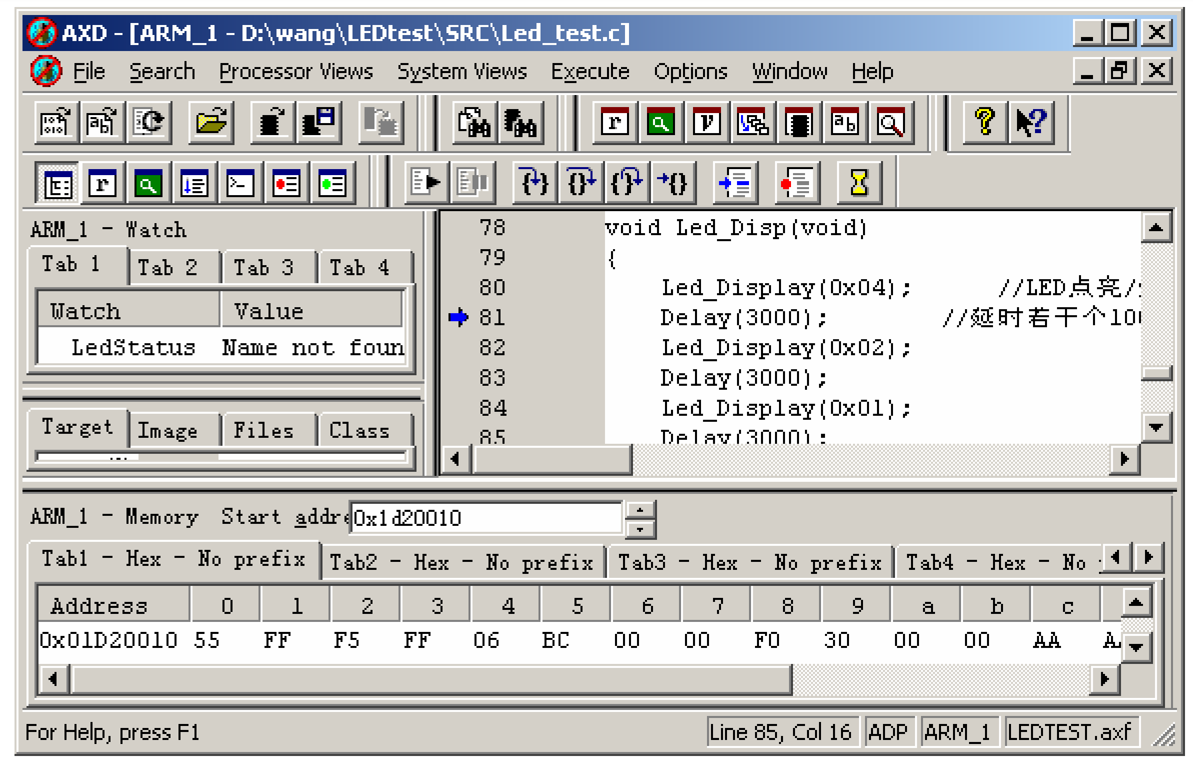


图 5-1-15 调试 LED

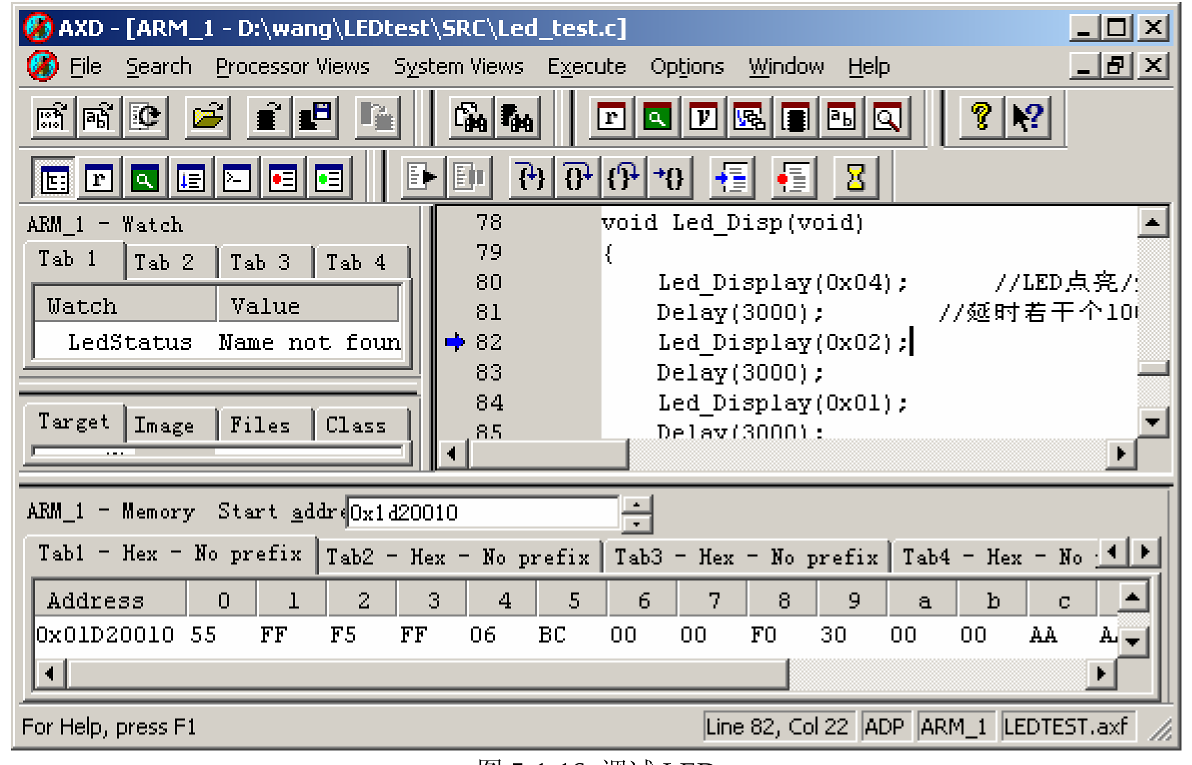


图 5-1-16 调试 LED

单击”Step In”再次调用 Led_Display()函数，但此次 LedStatus 的值是 0x02（在 watch 窗口中能看到变化）：

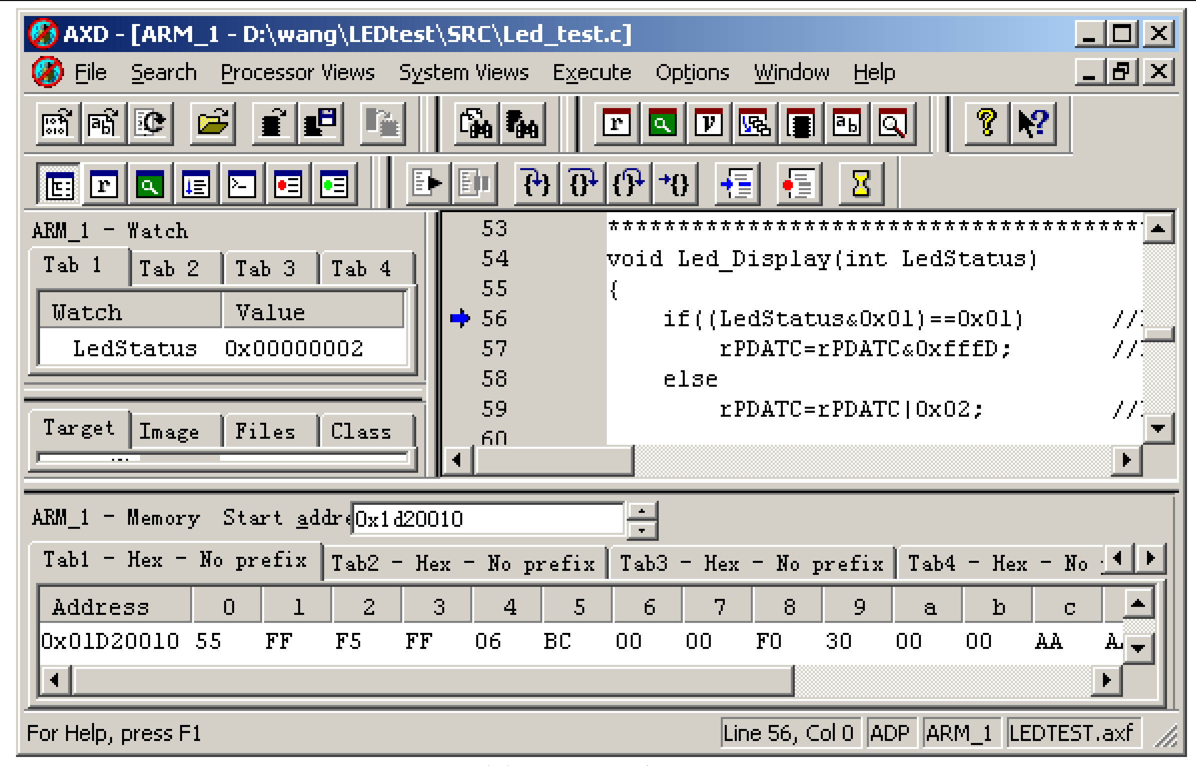


图 5-1-17 调试 LED

仍然单击”Step In”运行 Led_Display()函数，运行过程中可以看到第二个灯熄灭，第三个灯点亮，rPDATC 寄存器的低八位的值由 06 变为 02 又变为 0A；

用前面的方法继续运行代码，第三次调用 Led_Display()函数的过程中可以看到第一个灯熄灭，第二个灯点亮，rPDATC 寄存器的低八位的值由 0A 变为 08 又变为 0C。

Led_Disp()函数运行完后跳回到主函数，由于执行条件是 while(1)，所以开始第二次调用 Led_Disp()函数，程序的执行情况与与前面的一样。

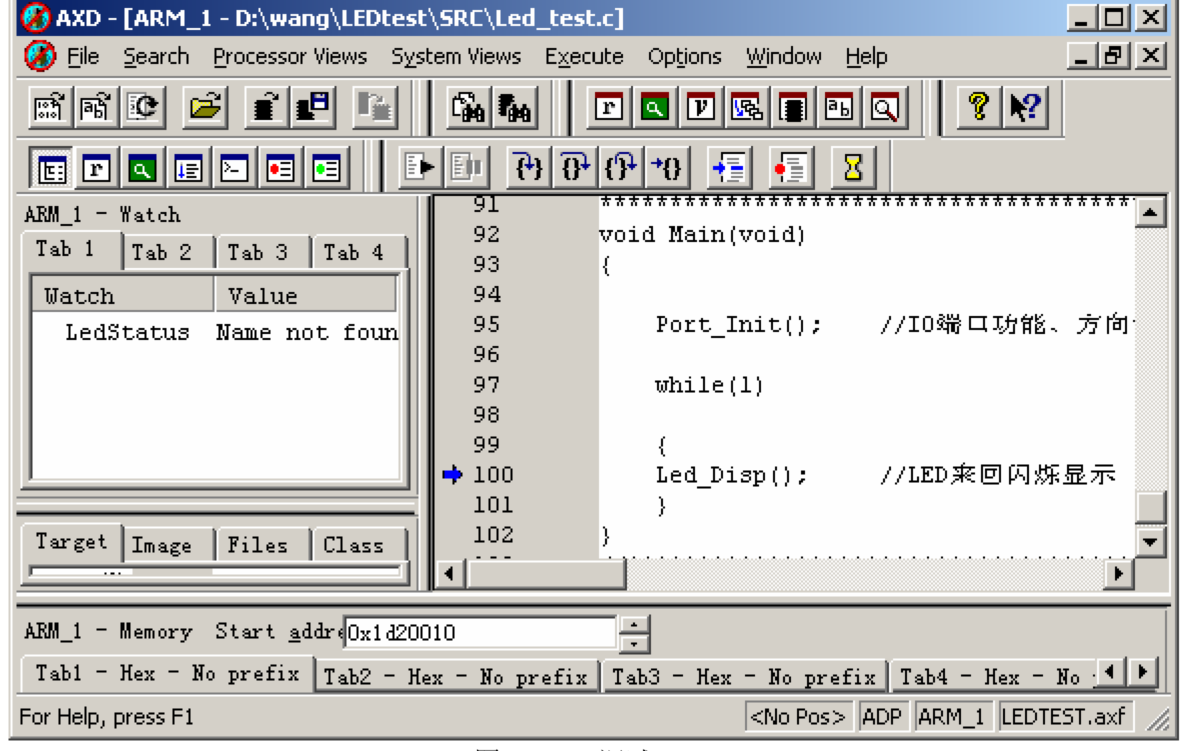


图 5-1-18 调试 LED

单击"Go"全速运行代码（单击" Stop" 停止），可以看到三个 LED 灯在不停地闪烁。

第二节 IIC 总线测试测试

1. IIC 总线简介

IIC BUS(Inter Integrated Circuit BUS 内部集成电路总线)是由 Philips 公司推出的双向、二线制、同步串行总线，是具备总线仲裁和高低速设备同步等功能的高性能多主机总线，应用极为广泛。

IIC 总线使用两根信号线来进行数据传输，一根是串行数据线(SDA)，另一根是串行时钟线(SCL)。它允许若干兼容器件(如存储器、A/D 和 D/A 转换器、以及 LED 驱动器、LCD 控制器等)共享总线。总线上所有器件要依靠 SDA 发送的地址信号寻址，不需要片选线。任何时刻总线只能由一个主器件控制，各从器件在总线空闲时启动数据传送，由 II C 总线仲裁来决定哪个主器件控制总线。

2. IIC 总线数据及应答 ACK 信号传送

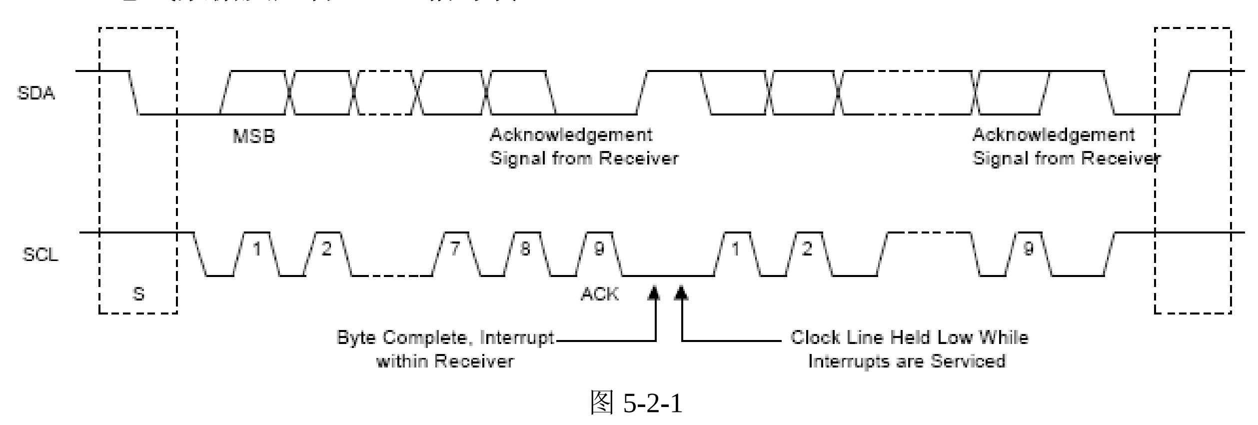


图 5-2-1

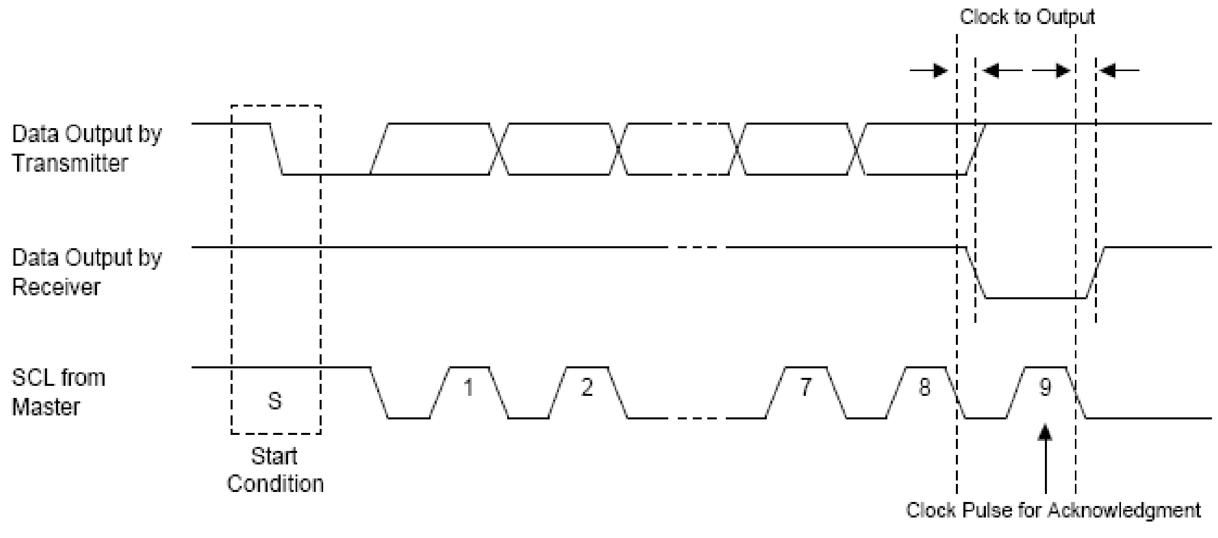


图 5-2-2

3. IIC 总线工作方式

1.〉主 / 发送模式流程图

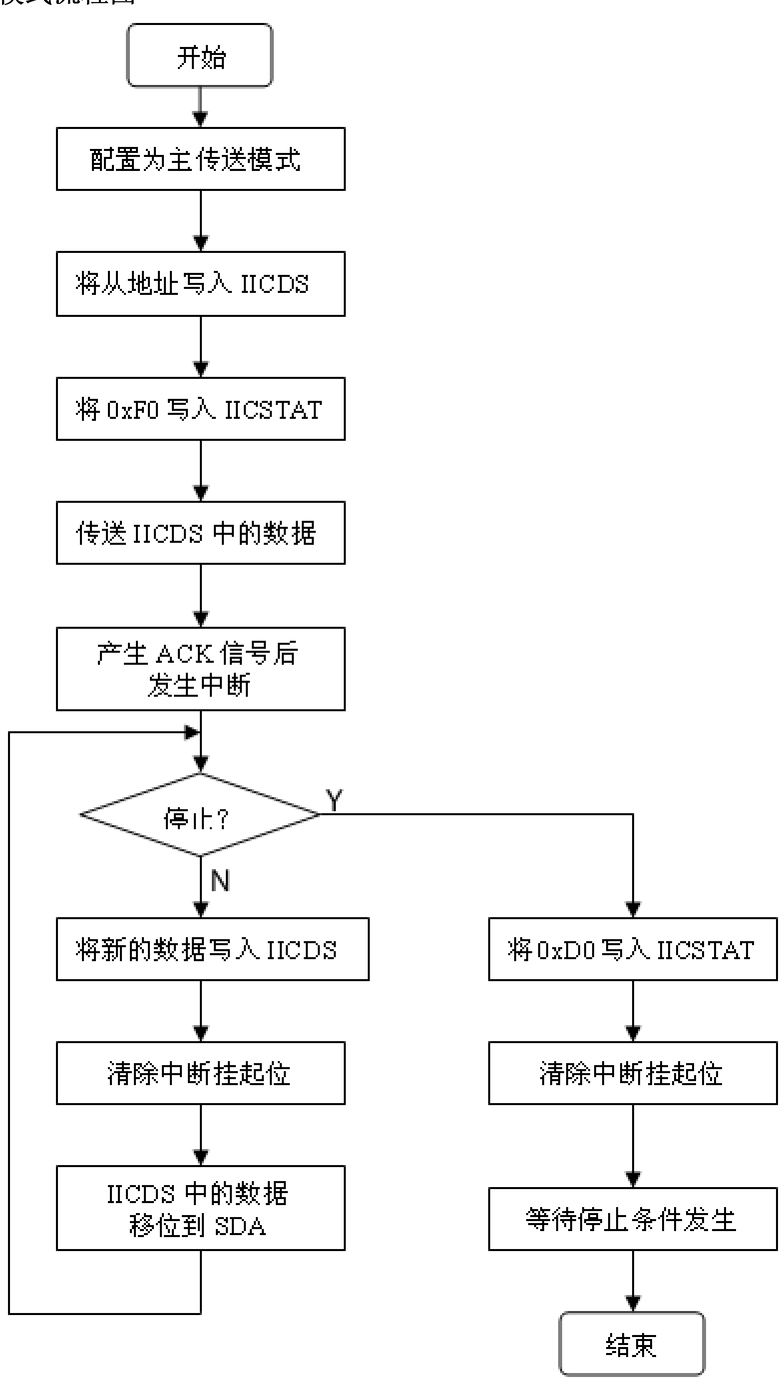


图 5-2-3

2.〉主／接收模式流程图

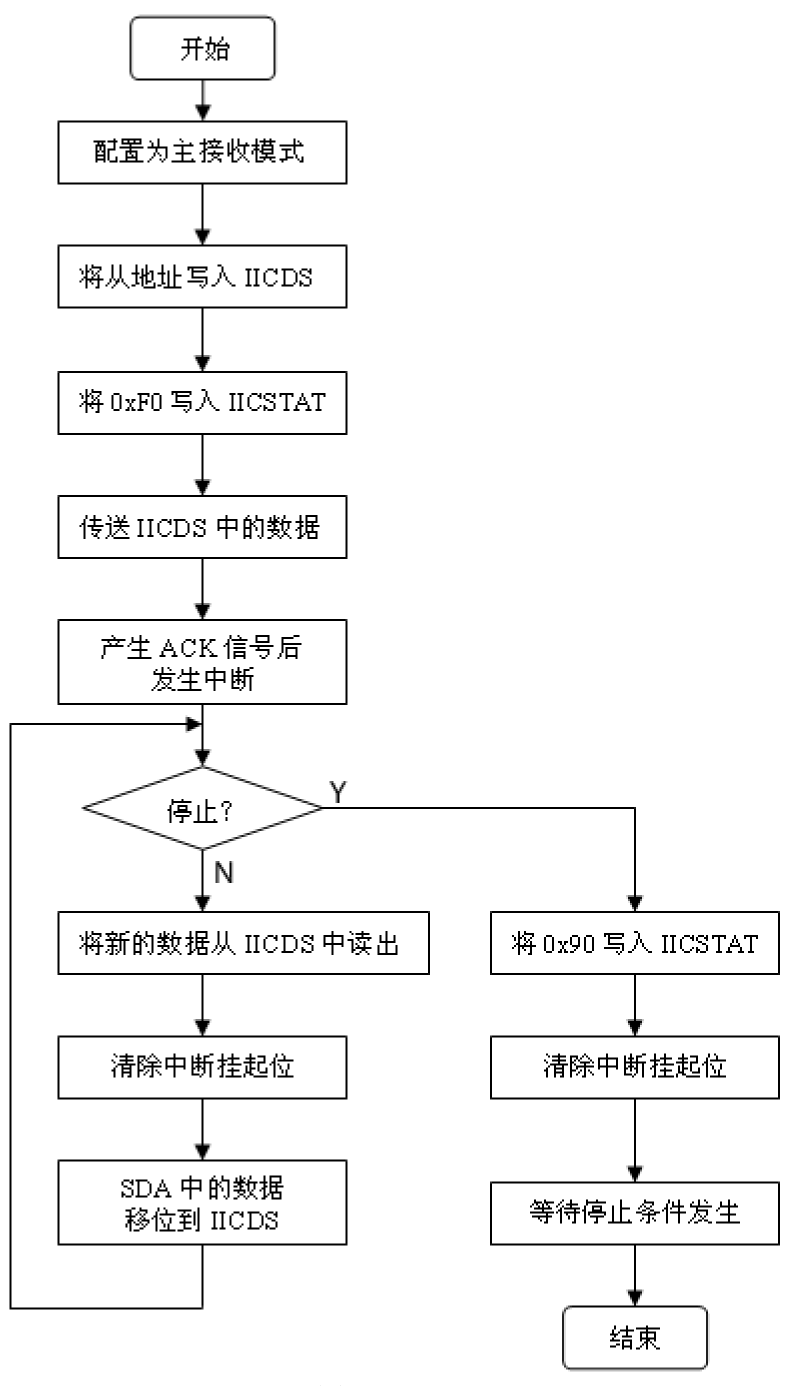


图 5-2-4

3.〉从 / 发送模式流程图

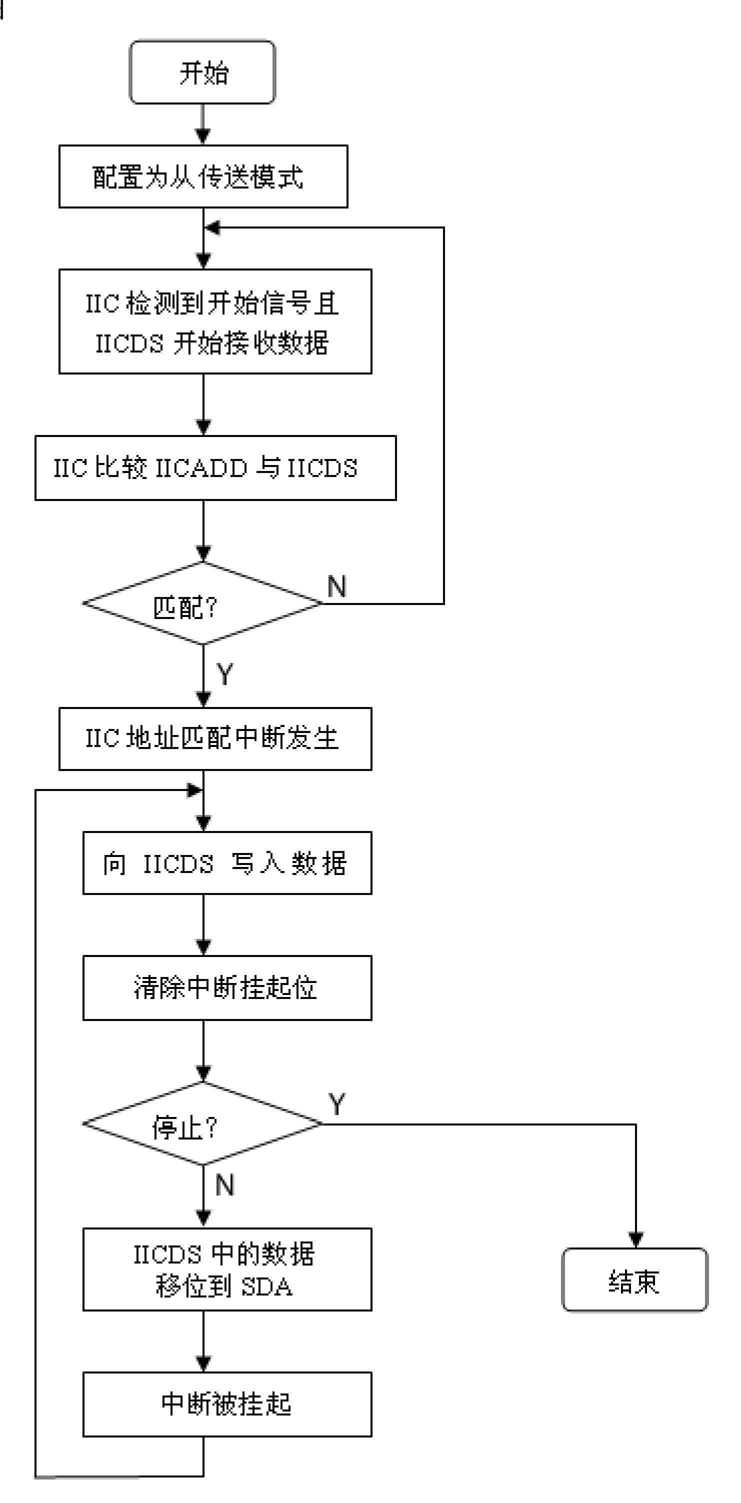


图 5-2-5

4 。〉从 / 接收模式流程图

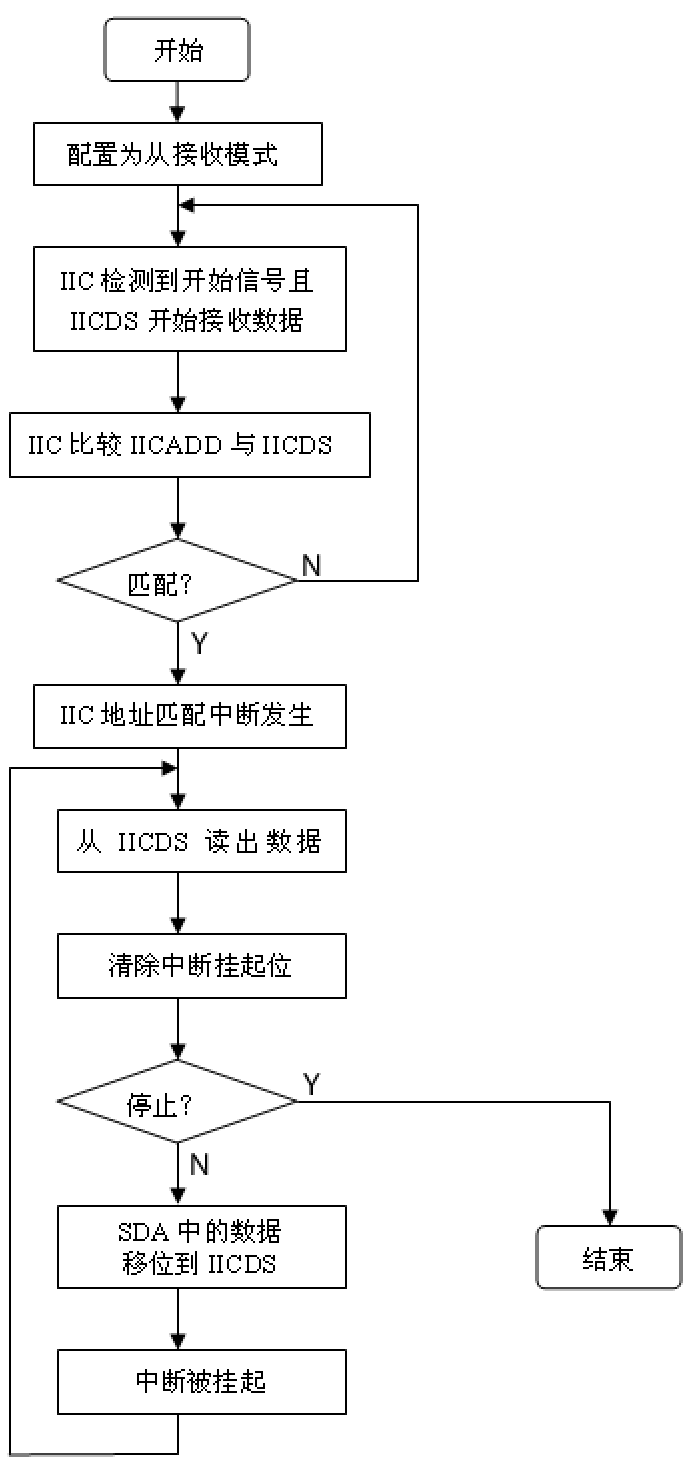


图 5-2-6

4. IIC 总线接口专用寄存器

1。〉 多主 IIC 总线控制寄存器（IICCON）

IICCON　　地址：0x01D60000　　R/W　　初始值：0000_xxxx

	位	描述
Acknowledge enable	[7]	应答允许位 0=禁止，　1=允许 发送模式时，IICSDA在ACK时释放 接收模式时，IICSDA在ACK时为低电平
Tx clock source selection	[6]	IIC总线传送时钟源预分频值选择位 0=IICCLK = f_{MCLK} /16　　1=IICCLK=$_{MCLK}$ /512
Tx/Rx Interrupt enable	[5]	IIC-总线 Tx/Rx中断使能/禁止位 0=禁止中断　　1=允许中断
Interrupt Pending flag	[4]	IIC-总线 Tx/Rx中断挂起标志位 0 = 1) 没有中断(读时) 2) 清除挂起条件并恢复操作(写时) 1 = 1) 中断挂起 (读时) 2) 无操作N/A (写时)
Transmit clock value	[3:0]	IIC传送时钟预分频值，传送时钟频率由这4位预分频值决定， 公式为：Tx clock = IICCLK/(IICCON[3:0]+1)

2 。〉多主 IIC 总线控制 / 状态寄存器（IICSTAT）

IICSTAT　　地址：0x01D60004　　R/W　　初始值：0000_0000

位名称	位	描述
Mode selection	[7:6]	IIC-总线主/从 Tx/Rx模式选择位 00=从接收模式　01=从发送模式 10=主接收模式　11=主发送模式
Busy signal status/ START STOP condition	[5]	IIC-总线忙信号状态位 0 = 1) IIC总线不忙(读时) 2) IIC总线STOP信号发生(写时) 1 = 1) IIC总线忙(读时) 2) IIC总线 (写时) 在 START 信号发生后 IICDS 中的数据就自动传输
Serial output enable	[4]	IIC总线数据输出使能/禁止位 0=禁止 Rx/Tx　1=允许 Rx/Tx
Arbitration status flag	[3]	IIC总线仲裁过程状态标志位 0 =总线仲裁成功　1=总线仲裁失败

Address-as-slave status flag	[2]	IIC总线从地址状态标志位 0=检测到 START/STOP 清除 1=接收到的从地址匹配 IICADD 中的地址值
Address zero status flag	[1]	IIC总线地址0状态标志位 0=检测到 START/STOP 清除　1=接收到的从地址是 00000000b
Last-received bit status flag	[0]	IIC总线上次接收位状态标志位 0=上次接收位是0(收到ACK) 1=上次接收位是 1(未收到 ACK)

3.〉 多主 IIC 总线地址寄存器（IICADD）

IICADD　　地址：0x01D60008　　R/W　　初始值：xxxx_xxxx

位名称	位	描述
Slave address	[7:0]	当IICSTAT中的串行输出使能位是0时，IICADD为写允许 IICADD的值可以在任意时候被读取，不管当前输出使能位(IICSTAT)的设置 从地址=[7:1]　传送方向指示位=[0]

4.〉 多主 IIC 总线发送 / 接收数据移位寄存器（IICDS）

IICDS　　地址：0x01D6000C　　R/W　　初始值：xxxx_xxxx

位名称	位	描述
Data shift	[7:0]	当IICSTAT中的串行输出使能位是1时，IICDS为写允许 IICDS的值可以在任意时候被读取，不管当前输出使能位(IICSTAT)的设置

5. 测试程序代码

1.〉 I/O 端口 IIC 功能设置

由原理图知 PF0 口和 PF1 口分别作为 IIC 总线的 SCL 线和 SDA 线与 KS24C080 相连，端口设置语句为：

```
rPCONF |=0xa;    //PF0:IICSCL, PF1:IICSDA
rPUPF |=0x3;     //上拉电阻禁止
```

2 。〉IIC 中断使能

定义中断处理程序入口，并且使能中断：

```
pISR_IIC=(unsigned)IicInt;   //将 IIC 中断处理程序指针指向 IicInt
rINTCON=0x5;                 //使能中断
rINTMOD=0x0;
rINTMSK=~(BIT_GLOBAL|BIT_IIC);
```

3.〉配置 IIC 总线

```
rIICCON=(1<<7)|(0<<6)|(1<<5)|(0xf);    //使能 ACK 的产生, IICCLK = fMCLK /16,  使能发送/接收中断,
                                       //清除中断挂起位，发送时钟频率= IICCLK / (15+1)
rIICADD=0x10;     //设置 S3C44B0X 从地址
rIICSTAT=0x10;    //使能 Rx/Tx
```

4.〉向 KS24080 中写入数据

```
#define WRDATA          (1)
#define POLLACK         (2)
#define RDDATA          (3)
#define SETRDADDR       (4)

#define IICBUFSIZE      0x20

void Wr24C080(U32 slvAddr,U32 addr,U8 data)
{
        _iicMode=WRDATA;        //写数据模式
        _iicPt=0;
        _iicData[0]=(U8)addr;       //字节写入模式，数据格式参考 KS24C080 资料
        _iicData[1]=data;
        _iicDataCount=2;

        rIICDS=slvAddr; //0xa0
        rIICSTAT=0xf0;                  //主发送模式，总线 STAR 信号产生，使能 Rx/Tx
        while(_iicDataCount!=-1);
        _iicMode=POLLACK;

        while(1)
        {
        rIICDS=slvAddr;
        _iicStatus=0x100;
        rIICSTAT=0xf0;              //主发送模式，总线 STAR 信号产生，使能 Rx/Tx
        rIICCON=0xaf;             //恢复 IIC  操作
        while(_iicStatus==0x100);
        if(!(_iicStatus&0x1))
                break;                  //  成功接收到 ACK
        }
        rIICSTAT=0xd0;            //产生停止条件
        rIICCON=0xaf;             //恢复 IIC  操作
        Delay(1);                 //等待，直到停止条件起作用
        //写入成功
}
```

5.〉从 KS24C080 中读出数据

```
void Rd24C040(U32 slvAddr,U32 addr,U8 *data)
{
        _iicMode=SETRDADDR;      //设置读地址模式
        _iicPt=0;
        _iicData[0]=(U8)addr;           //字节读出模式，数据格式参考 KS24C080 资料
        _iicDataCount=1;

          rIICDS=slvAddr;
```

```
        rIICSTAT=0xf0;              //主发送模式，总线 STAR 信号产生，使能 Rx/Tx
        while(_iicDataCount!=-1);
       _iicMode=RDDATA;         //读数据模式
        iicPt=0;
       _iicDataCount=1;
        rIICDS=slvAddr;
        rIICSTAT=0xb0;             //主接收模式，IIC 总线忙，使能 Rx/Tx
        rIICCON=0xaf;              //恢复 IIC  操作
        while(_iicDataCount!=-1);
       *data=_iicData[1];
}
```

6.〉 IIC 中断处理程序

```
void __irq IicInt(void)
{
        U32 iicSt,i;
        rI_ISPC=BIT_IIC;          //清除中断挂起位
        iicSt=rIICSTAT;           //读入 IIC 总线当前状态，以便进行各种错误处理
        if(iicSt&0x8){}           //  如果总线仲裁失败
        if(iicSt&0x4){}           //  如果某个从器件地址与 IICADD 匹配
        if(iicSt&0x2){}           //  如果从地址为 0000000b
        if(iicSt&0x1){}           //  如果没有收到 ACK 信号

        switch(_iicMode)          //根据当前操作模式进行数据处理
        {
        case POLLACK:            //等待 ACK 模式
                _iicStatus=iicSt   //读入 IICSTAT，第 0 位表示是否接收到 ACK
                break;

        case WRDATA:                        //写数据模式
                if((_iicDataCount--)==0)    //如果数据写完
                {
                rIICSTAT=0xd0;              //主发送模式，STOP 信号产生
                rIICCON=0xaf;               //恢复 IIC 操作
                Delay(1);                   //等待，直到停止条件起作用
                break;
                }
                rIICDS=_iicData[_iicPt++];
                for(i=0;i<10;i++);          //在 IICSCL 上升沿之前有一个建立时间
                rIICCON=0xaf;               //恢复 IIC 操作
                break;

        case RDDATA:                          //读数据模式
                if((_iicDataCount--)==0)      //只读取 1 字节数据
                {
```

```
            _iicData[_iicPt++]=rIICDS;
            rIICSTAT=0x90;              //主接收模式，IIC 总线空闲
            rIICCON=0xaf;               //恢复 IIC 操作
            Delay(1);                   //等待停止条件起作用
            break;
            }
            _iicData[_iicPt++]=rIICDS;    //还未读完所有数据最后一个字节不能产生 ACK
            if((_iicDataCount)==0)       //如果读完了所有数据（最后一个字节）
            rIICCON=0x2f;               //产生  NOACK，恢复 IIC 操作
            else
            rIICCON=0xaf;               //产生  ACK，恢复 IIC 操作
            break;

        case SETRDADDR:                 //设置读地址模式
            if((_iicDataCount--)==0)
            {
            break;                      //由于 IICCON[4]置位，IIC 操作停止
            }
            rIICDS=_iicData[_iicPt++];
            for(i=0;i<10;i++);          //在 IICSCL 上升沿之前有一个建立时间
            rIICCON=0xaf;               //恢复 IIC 操作
            break;

        default:
            break;
      }
    }
```

6．运行结果

该程序首先向 KS24C080 中写入 256 个字符，然后再读出这 256 个字符（共 256 个字节），并将他们送至串口输出，运行结果如下图所示。

```
*----------------Begin to Start IIC test,OK? (Y/N)-----------------*

Enter the 'Y' or 'y' to test:

Write test data into KS24C080
Read test data from KS24C080
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f
80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f
90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f
a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af
b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf
d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df
e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff
```

图 5-2-7

第三节看门狗定时器测试

1．看门狗定时器简介

看门狗定时器根据程序在指定时间间隔内是否进行相应的操作来判断程序运行是否出错，是保证嵌入式软件长期、可靠和稳定运行的有效措施之一。S3C44B0X 看门狗定时器它具有以下特征：带中断请求的普通16 位定时器模式；当定时器计数值达到 0 时，内部复位信号被激活 128 MCLK 周期。

2． 看门狗定时器功能框图

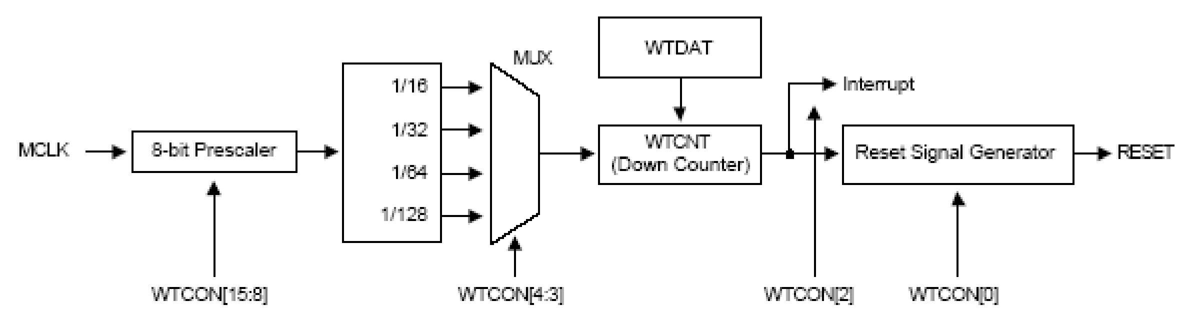


图 5-3-1

定时器时钟周期 t_watchdog = 1/MCLK/(预分频值+1)/分频系数

3．看门狗定时器特殊功能寄存器

看门狗定时器控制寄存器（WTCON）

WTCON 地址：0x01D30000 R/W 初始值 0x8021

位名称	位	描述
Prescaler value	[15:8]	预分频值：0~ (2^8 -1)
Reserved	[7:6]	保留，在正常状态下这两位必须为 00
watchdog timer enable/disable	[5]	看门狗定时器允许位 0=禁止 1=允许
Clock select	[4:3]	确定时钟除数因子： 00: 1/16 01: 1/32 10: 1/64 11: 1/128
Interrupt enable/disable	[2]	看门狗中断允许位 0=禁止中断 1=允许中断
Reserved	[1]	保留，在正常状态下该位必须为 0
Reset enable/disable	[0]	看门狗复位信号输出允许位 0=禁止 1=允许

看门狗定时器数据寄存器（WTDAT）

WTDAT 地址：0x01D30004 R/W 初始值 0x8000

位名称	位	描述
Count reload value	[15:0]	看门狗定时器重载的计数值

看门狗定时器计数寄存器（WTCNT）

WTCNT 地址：0x01D30008 R/W 初始值 0x8000

位名称	位	描述
Count value	[15:0]	看门狗定时器的当前计数值

测试程序代码

```
void Test_WDTimer(void)
{
	Uart_Printf("WatchDog Timer Test:\n\n");
	rINTMSK=~(BIT_GLOBAL|BIT_WDT);   //使能中断
	pISR_WDT=(unsigned)Wdt_Int;     //将 WDT 中断处理程序指针指向 Wdt_Int
	isWdtInt=0;

	rWTCON=((MCLK/1000000-1)<<8)|(3<<3)|(1<<2); //预分频值= MCLK/1000000-1,除数因子=1/128,
	                                            //使能看门狗中断
	rWTDAT=8448/2;
	rWTCNT=8448/2;
	rWTCON=rWTCON|(1<<5);     //  使能看门狗定时器
	while(isWdtInt!=10);

	rWTCON=((MCLK/1000000-1)<<8)|(3<<3)|(1);   //预分频值= MCLK/1000000-1,除数因子=1/128,
	                                           //禁止看门狗中断，使能看门狗复位输出
	Uart_Printf("\nI will restart after 3 sec!!!\n");
	rWTCNT=8448*3;
	rWTCON=rWTCON|(1<<5);     //使能看门狗定时器
	while(1);
	rINTMSK|=BIT_GLOBAL;     //所有中断禁止
}

void __irq Wdt_Int(void)     //中断处理子程序
{
	rI_ISPC=BIT_WDT;   //清除 pending 位
	Uart_Printf("%d ",++isWdtInt);    串口输出数字
}
```

运行结果

运行该测试程序，首先串口输出 1~10 十个数字，等待 3 秒后系统自动重启。

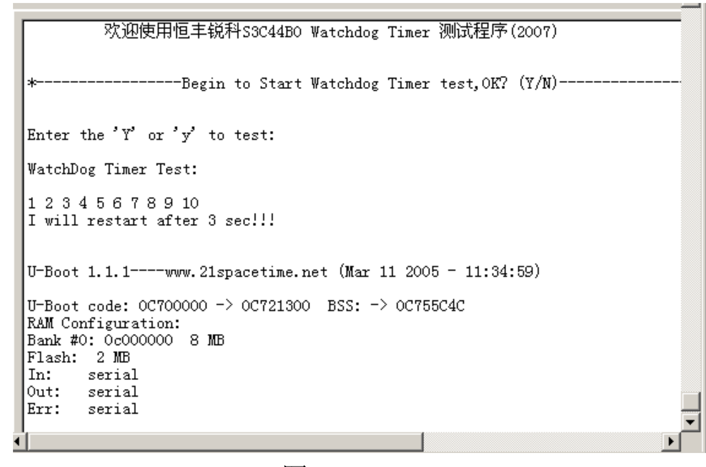


图 5-3-2

第四节实时时钟测试

1. S3C44B0 RTC 概述

实时时钟（Real Time Clock）是一种能提供日历 / 时钟及数据存储等功能的专用集成电路，具有计时准确、体积小、功耗低等特点，常用作各种计算机系统的时钟信号源及参数设置存储电路。

系统电源关闭时，RTC 单元可以通过备用电池供电而继续工作。RTC 可通过 STRB/LDRB 指令向 CPU 传送 8 位 BCD 数据，传送的数据包括秒、分、时、日、星期、月、年。RTC 单元由一个外部 32.768kHZ 晶振提供时钟源并且具有报警功能。S3C44B0 实时时钟（RTC）有如下特性：

★ BCD 数据：秒、分、时、日、星期、月、年；
★ 闰年产生器；
★ 报警功能：报警中断或从掉电模式唤醒；
★ 独立的电源端（VDDRTC）；
★ 支持毫秒节拍时间中断作为 RTOS 内核的时间节拍；
★ 循环复位功能。

2. S3C44B0 RTC 功能框图

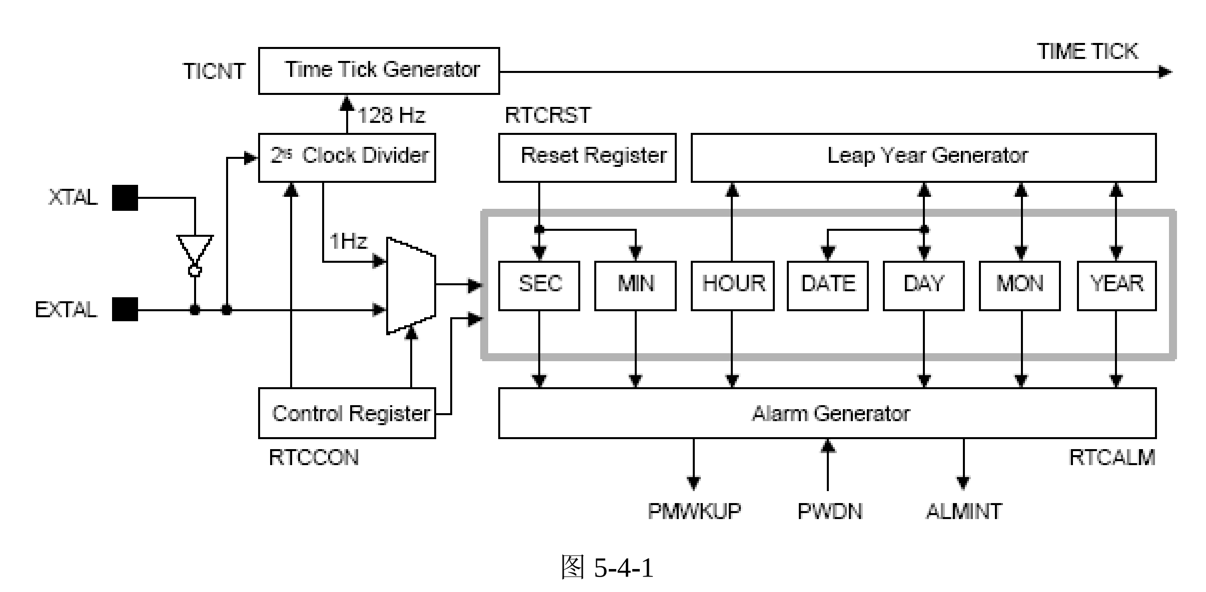


图 5-4-1

3. S3C44B0 RTC 特殊功能寄存器

1. RTC 控制寄存器（RTCCON）

RTCCON 有 4 位，CLKRET、CNTSEL 和 CLKSEL 用于测试，RTCEN 用于控制对 BCD 寄存器的读 / 写使能。RTCEN 位可以控制 CPU 和 RTC 之间的所有接口，因此在 RTC 控制程序中应将该位置 1，以保证系统复位后能够读 / 写数据。在断电前 RTCEN 位应清 0 ，以防止对 RTC 寄存器的意外写入。

RTCCON　地址：0x01D70040(小端) / 0x01D70043(大端)　R/W(字节)　初始值 0x0

位名称	位	描述
CLKRST	[3]	RTC时钟计数复位 0=不复位　1=复位
CNTSEL	[2]	BCD计数选择位 0=组合BCD计数器 1=保留(分开的 BCD 计数器

CLKSEL	[1]	BCD时钟选择位 0=XTAL 1/2^{15} divided clock 1 = 保留 (XTAL clock only for test)
RTCEN	[0]	RTC读 / 写允许位 0 =禁止 1 = 允许

2. RTC 报警控制寄存器（RTCALM）

RTCALM决定报警使能和报警时间。注意在掉电模式下，RTCALM寄存器通过ALMINT和PMWKUP产生报警信号；在正常工作模式下，只通过ALMINT产生报警信号。

RTCALM 地址：0x01D70050(小端) / 0x01D70053(大端) R/W(字节) 初始值 0x0

位名称	位	描述
Reserved	[7]	保留
ALMEN	[6]	Alarm全局允许 0 = 禁止 1 = 使能
YEAREN	[5]	年Alarm允许 0 = 禁止 1 = 使能
MONREN	[4]	月Alarm允许 0 = 禁止 1 = 使能
DAYEN	[3]	天Alarm允许 0 = 禁止 1 = 使能
HOUREN	[2]	时Alarm允许 0 = 禁止 1 = 使能
MINEN	[1]	分Alarm允许 0 = 禁止 1 = 使能
SECEN	[0]	秒Alarm允许 0 = 禁止 1 = 使能

3. RTC 循环复位寄存器（RTCRST）

RTCRST 地址：0x01D7006C(小端) / 0x01D7006F(大端) R/W(字节) 初始值 0x0

位名称	位	描述
SRSTEN	[3]	循环秒复位允许 0 = 禁止 1 = 使能
SECCR	[2:0]	产生秒进位的循环边界 011 = 超过 30s 100 = 超过 40s 101 = 超过 50s

4 .TICK TIME 计数寄存器（TICNT）

TICNT 地址：0x01D7008C(小端) / 0x01D7008F(大端) R/W(字节) 初始值 0x00

位名称	位	描述
TICK INT ENABLE	[7]	节拍时间中断允许 0 = 禁止 1 = 使能
TICK TIME COUNT	[6:0]	节拍时间计数值(1-127) 该计数值内部递减，不能读它的实时值

ALMSEC~ALMYEAR 及 BCDSEC~BCDYEAR 略

4.测试程序代码

1 .RTC 初始化

```
void Rtc_Init(void)
{
	rRTCCON = 0x01;              //使能 R/W, BCD 计数时钟选择 1/32768,
	                             //选择组合 BCD 计数, RTC 时钟计数不复位
	rBCDYEAR = TESTYEAR;         //设置各 BCD 数据寄存器的初值
```

```
        rBCDMON    = TESTMONTH;
        rBCDDAY    = TESTDAY;
        rBCDDATE = TESTDATE;
        rBCDHOUR = TESTHOUR;
        rBCDMIN    = TESTMIN;
        rBCDSEC    = TESTSEC;
}
```

2. RTC 读取显示

```
void Display_Rtc(void)
{
        Uart_Printf("This test should be excuted once RTC test for RTC initialization\n");
        Rtc_Init();
        pISR_TICK=(unsigned)Rtc_display; //将中断处理程序指针指向 Rtc_display
        rRTCCON = 0x01;                          // R/W  使能
        sec_tick=1;
        rINTMSK=~(BIT_GLOBAL|BIT_TICK);
        rTICINT = 127+(1<<7);                    //开始
        Uart_Getch();
        rINTMSK |= (BIT_GLOBAL | BIT_TICK);
        rRTCCON=0x0;                             // R/W 禁止
}

void __irq Rtc_display(void)   //RTC 读取显示中断服务程序
{
        rI_ISPC=BIT_TICK;       //清除中断挂起位
        if(rBCDYEAR == 0x99)
        year = 0x1999;
else

        year = 0x2000 + rBCDYEAR;
        month=rBCDMON;
        day=rBCDDAY;
        weekday=rBCDDATE;
        hour=rBCDHOUR;
        min=rBCDMIN;
        sec=rBCDSEC;
Uart_Printf("\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b%4x,%2x,%2x,%s,%2x:%2x:%2x",year,month,day,
date[weekday],hour,min,sec);
}
```

3. RTC 报警中断

```
volatile int isRtcInt;
int Test_Rtc_Alarm(void)
{
        Uart_Printf("RTC Alarm Test for S3C44B0X\n");
        Rtc_Init();
```

```
        rRTCCON = 0x01;    // R/W 使能
        rALMYEAR=rBCDYEAR ; //设置各报警数据寄存器的值
        rALMMON =rBCDMON ;
        rALMDAY =rBCDDAY ;
        rALMHOUR=rBCDHOUR;
        rALMMIN =rBCDMIN ;
        if(rBCDSEC<0x30)
          rALMSEC =0x30;
        else
          rALMSEC =0x59;
        isRtcInt=0;
        pISR_RTC=(unsigned int)Rtc_Int;          //将中断处理程序指针指向 Rtc_Int
        rRTCALM=0x7f;                            //使能年、月、日、天、时、分、秒及全局 Alarm
        rINTMSK=~(BIT_GLOBAL|BIT_RTC); //允许 RTC 中断
        while(isRtcInt==0);
        rINTMSK|=BIT_GLOBAL;                     //  禁止所有中断
        rRTCCON = 0x0;       // R/W  禁止(为降低功耗)
        return 1;
}

void __irq Rtc_Int(void) //RTC 报警中断服务程序
{
        rI_ISPC=BIT_RTC;
        Uart_Printf("RTC Alarm Interrupt O.K.\n");
        isRtcInt=1;
}
```

4.RTC 节拍时间中断

```
void Test_Rtc_Tick(void)
{
        Uart_Printf("RTC Tick Interrupt for S3C44B0X\n");
        pISR_TICK=(unsigned)Rtc_Tick;
        sec_tick=1;
        rINTMSK=~(BIT_GLOBAL|BIT_TICK);
        rRTCCON=0x0;
        rTICINT = 127+(1<<7);
        Uart_Getch();
        rINTMSK |= (BIT_GLOBAL | BIT_TICK);
        rRTCCON=0x0;
 }
void __irq Rtc_Tick(void)
{
        rI_ISPC=BIT_TICK;
        Uart_Printf("\b\b\b\b\b\b\b%03d sec",sec_tick++);
}
```

5.运行结果

运行该程序，通过输入“0”、“ 1 ”、“ 2 ”来选择要测试的 RTC 功能，结果如下：

```
******************************************************************
          欢迎使用恒丰锐科S3C44B0 RTC 测试程序(2007)
*-------------------Begin to Start RTC test--------------------*
 0:RTC(display)     1:RTC(Alarm)        2:RTC(Tick)
Select the function to test:0
This test should be excuted once RTC test for RTC initialization
2000, 1, 1,SAT, 0: 0:16
```

图 5-4-2

```
******************************************************************
          欢迎使用恒丰锐科S3C44B0 RTC 测试程序(2007)
*-------------------Begin to Start RTC test--------------------*
 0:RTC(display)     1:RTC(Alarm)        2:RTC(Tick)
Select the function to test:1
RTC Alarm Test for S3C44B0X
RTC Alarm Interrupt O.K.
```

图 5-4-3

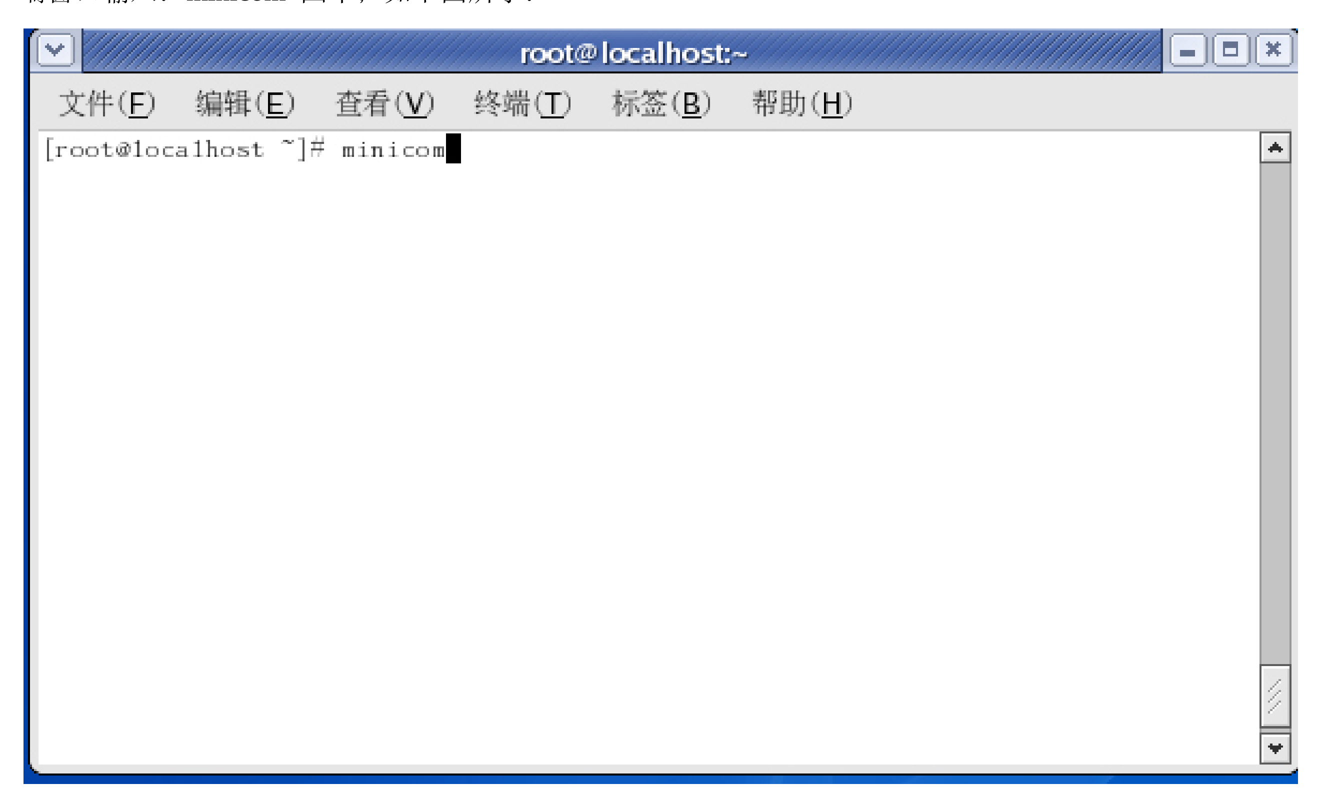


图 5-4-4

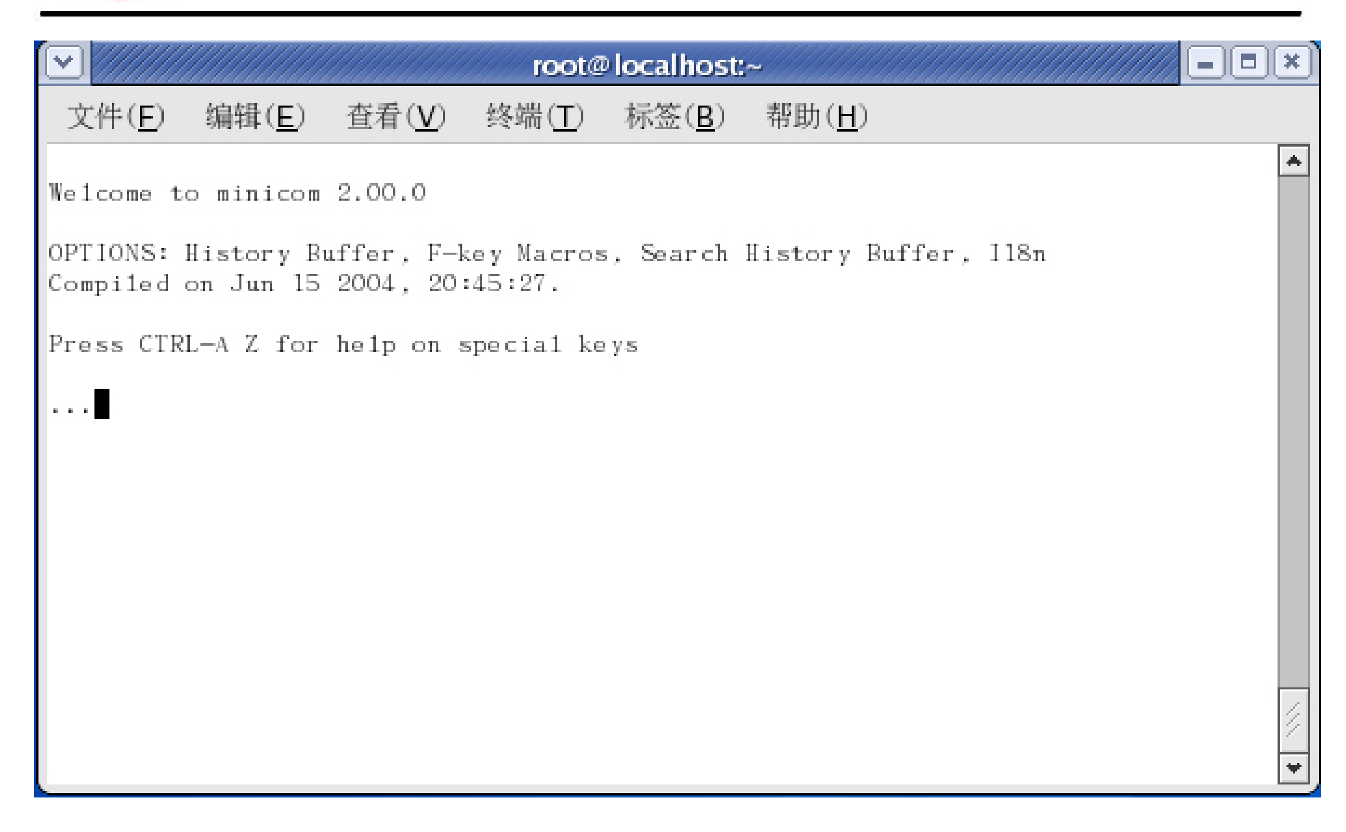
第六章在 Linux 下使用 minicom 调试 ARM7 开发板

我们在此使用的操作系统是 redhat Fedord3，其他版本的 linux，操作过程也很类似。minicom 是个通信程序，有点象共享软件 TELIX，但其源码可以自由获得，并能够运行于多数 Unix 系统。 它包括以下特性：自动重拨号的拨号目录， 对串行设备 UUCP 格式的 lock 文件的支持，独立的脚本语言解释器，文件捕获，多用户单独配置，等等。通常，minicom 从文件"minirc.dfl"中获取其缺省值。

我们在此不过多地介绍这个软件，只是熟悉怎么使用这个软件来下载程序到我们的开发板。首先在终端窗口输入：minicom 回车；如下图所示：



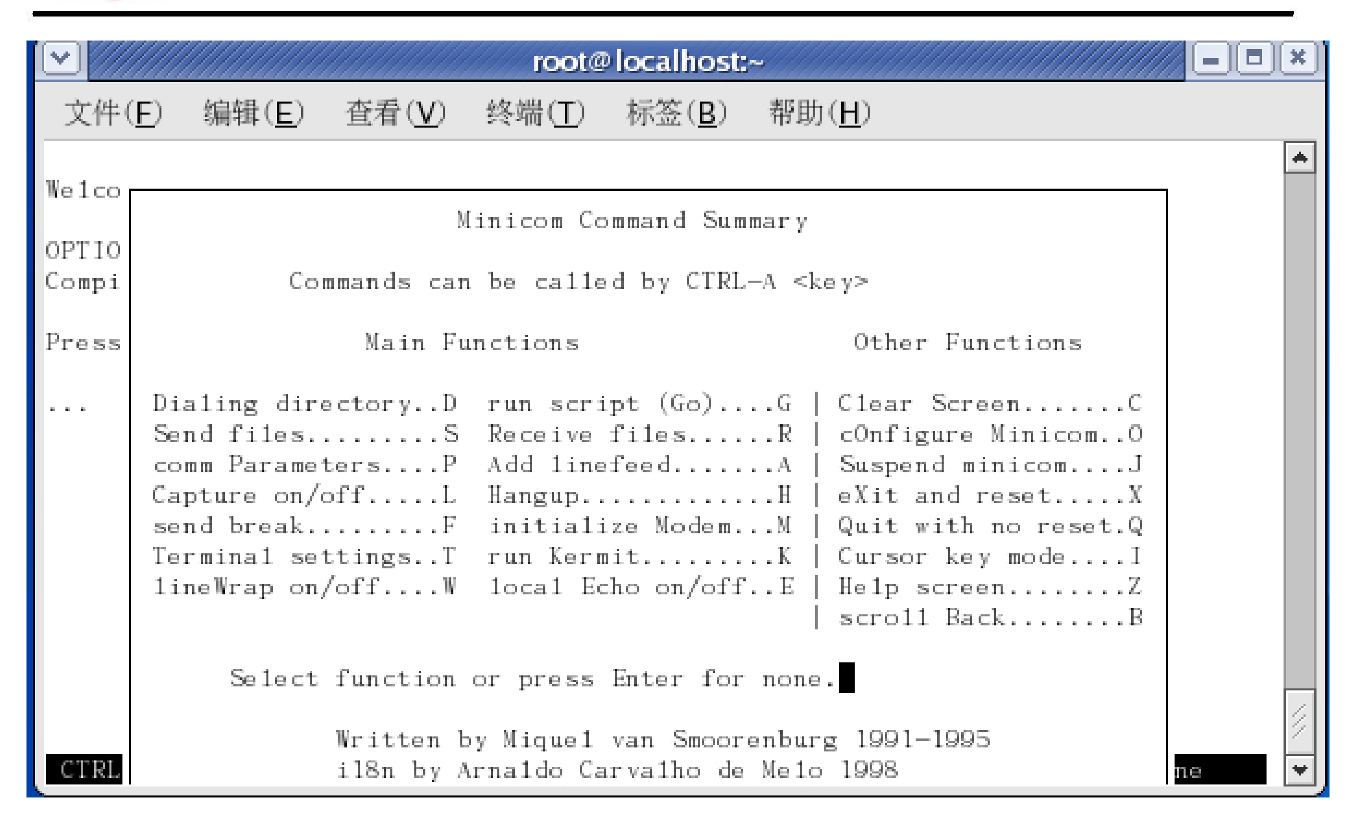
等 minicom 运行以后将显示如下图，



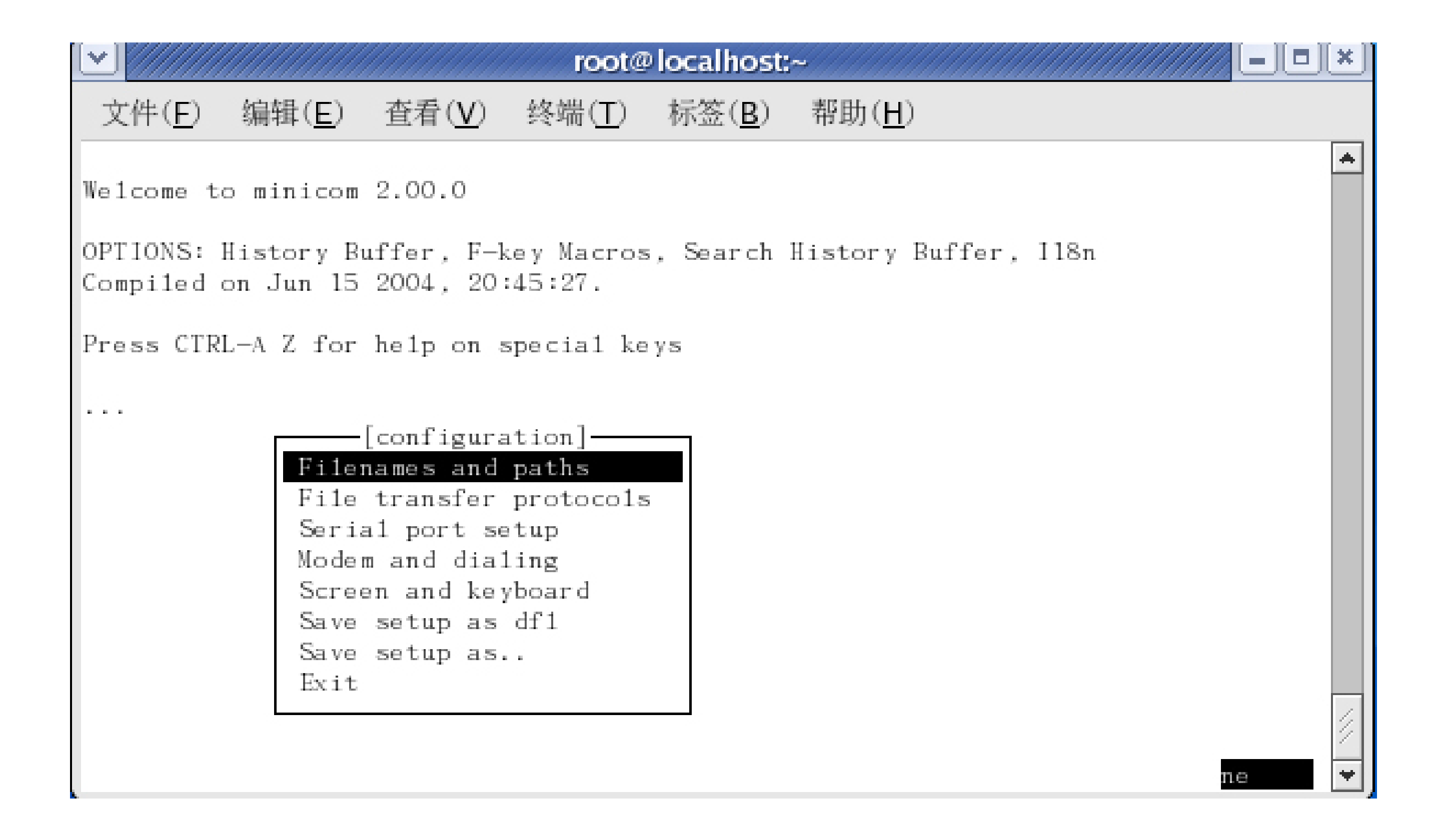
我们按 CTRL+A 键，显示如下



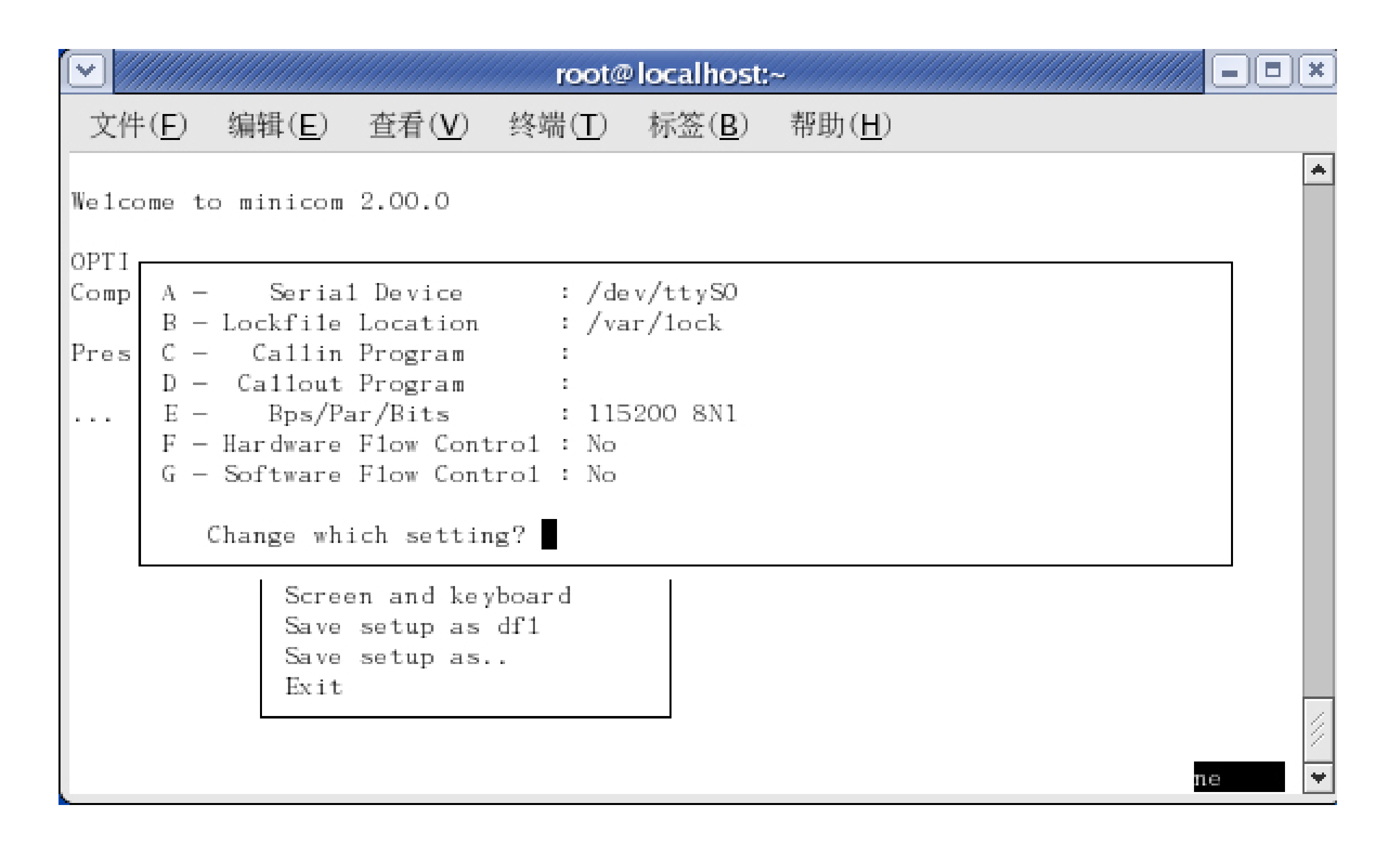
接下来，我们按字母 Z，将出现帮助和命令菜单，如下图所示，我们按照说明对应的字母来按键盘即可进入相应的设置选项，我们首先来设置我们的串口。



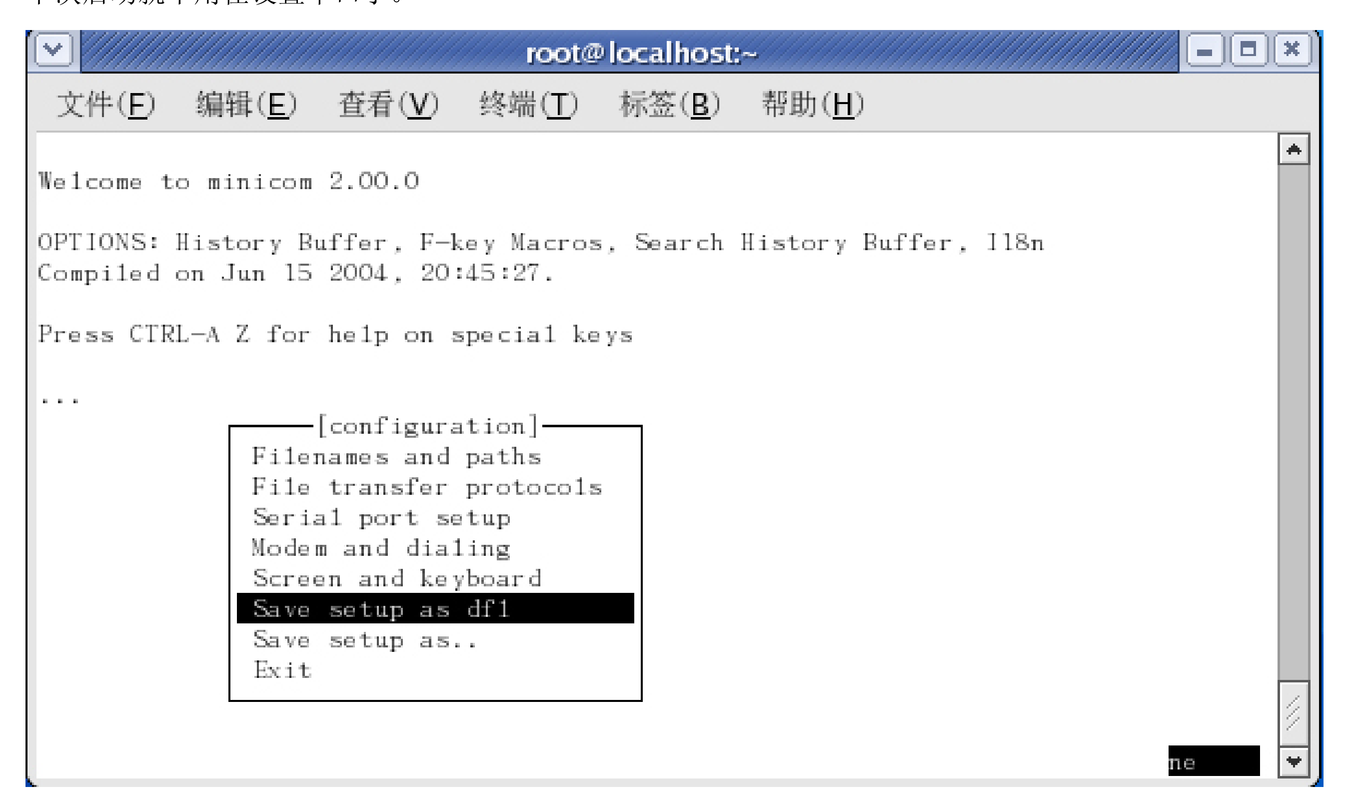
我们从键盘输入“O”，显示如下：



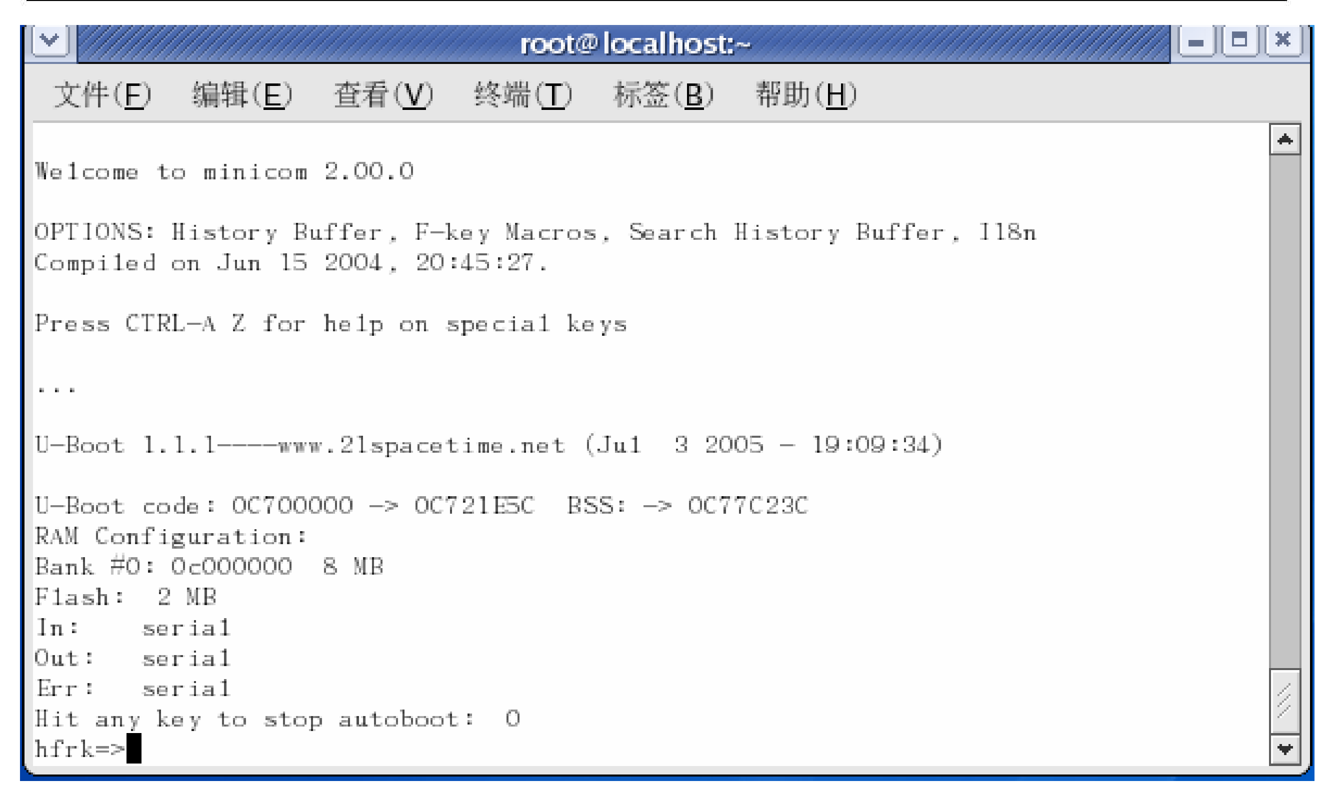
我们来配置串口，选择 Serial port setup，回车，程序进入串口设置，如下图所示，我们选择相应的串口和速率，回车退出。（com0 是/dev/ttyS0）



我们可以把我们刚才的设置保存为默认的设置，如下图，选择 save setup as df1,回车即保存了我们设置，下次启动就不用在设置串口了。

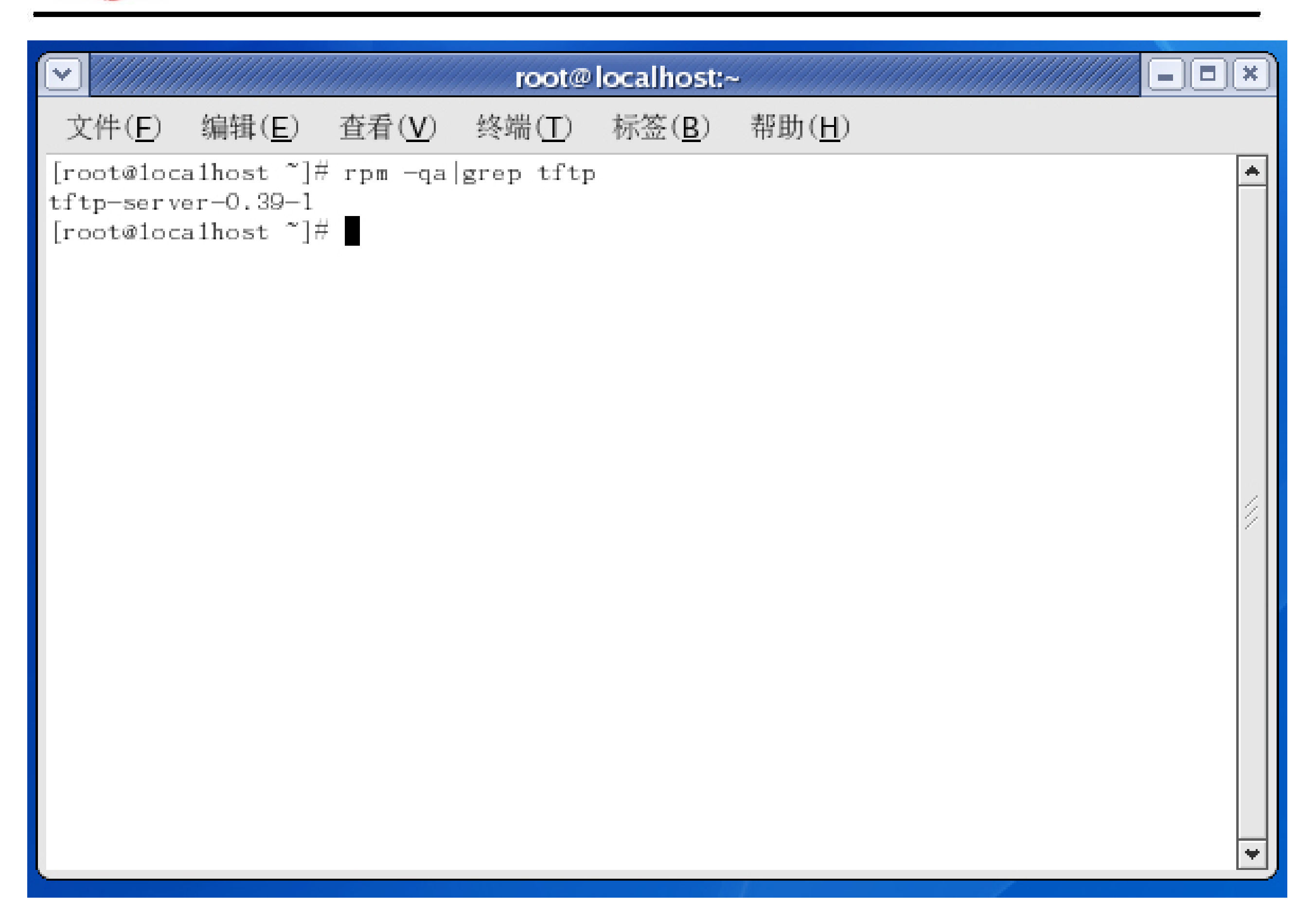


接下来选择 Exit 退出，我们启动开发板，就有信息输出，我们的 minicom 已经配置好了。



我们如果想退出 minicom，可以按 CTAL+A，然后按 Z 进入菜单，输入 Q，即可退出，如果输入 J，可以暂时退到命令提示符，在命令提示符下输入 fg 回车又可以重新回到 minicom 控制界面。

我们已经可以操作开发板了，下面说明如何在 linux 下配置 TFTP，首先用 rpm －qa | grep tftp 看一下 tftp 有没安装，没有的话，安装一下。 如下图所示：



假如已经安装好了，我们可以检查一下配置文件，培植文件位于/etc/xinetd.d/tftp

```
service tftp
{
        disable = no
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -u nobody -s /tftpboot
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

我们可以看到 tftp 指定的目录是/tftpboot，所以我们把要下载的文件拷贝到这个目录下就可以了，配置完成后，我们重新启动 tftp 服务，输入命令 service xinetd restart 因为 TFTP 服务受控与 xinetd，所以以启 xinetd 服务就可以了。



现在我们的 tftp 服务器已经设置完成，可以使用 tftp 和 minicom 下载程序了，但是有一点需要注意，如果你启动了防火墙，有可能不能正常下载，我们需要打来端口 69，在此我们也可以做如下选择，点击[应用程序]->[系统设置]->[安全级别]，显示如下图，因为我们的开发办接在 eth0 上，所以我们可以选择 eth0 为信任设备，点击[确定]即可，另外也可以在[其它端口：]的文本框里输入 69:udp 也可以。

安全级别配置
请为您的系统选择安全级别。
防火墙选项(F) SELinux (S)
安全级别: 启用防火墙
信任的服务: WWW (HTTP) FTP SSH Telnet 邮件(SMTP)
信任的设备: eth0 sit0
其它端口：(1029:tcp)
取消(C) 确定(O)

第七章 如何移植 uclinux2.6 内核

注：用户可以在串口控制台和 framebuffer 间切换，切换成 framebuffer 的时候，在 core application 菜单中要选择 minix-shell 而串口控制台要选择 sash shell.

注意：光盘配带的 uclinux 已经移植好了。所以不需要这些工作。在此只是让你了解过程。

第一节建立开发环境。

主机环境: redhat 9,内核：2.4.20。
交叉编译器: arm-elf-tools-20040427.
uclinux-dist 发行盘： uclinux-dist-20050311.tar.gz.

我们可以在网上下载这些资源在

www.uclinux.org 下载了 linux-2.6.9-uc0.patch；

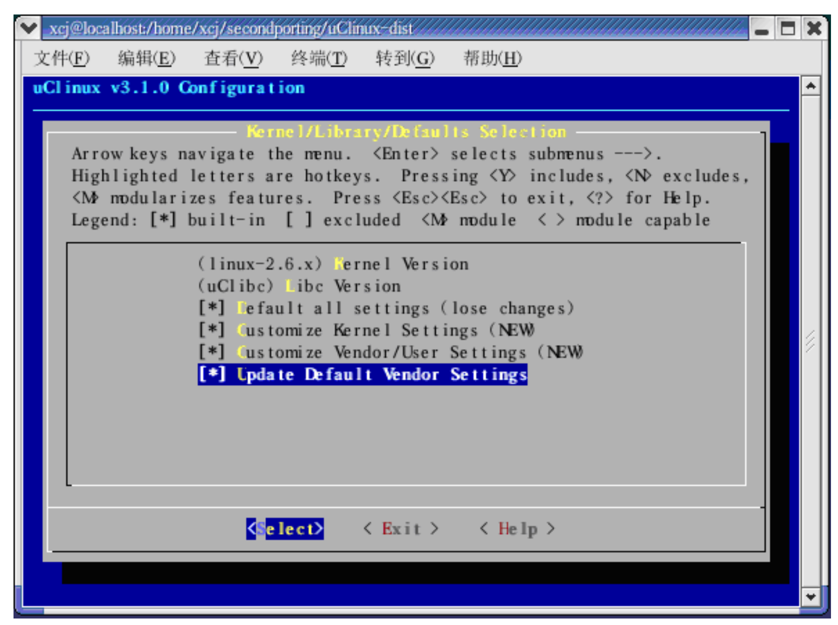
在 http://opensrc.sec.samsung.com 下载了 linux-2.6.9-uc0-hsc0 patch

在 www.kernel.org 下载了 linux-2.6.9.

好了，有了这些东东，我们就可以开始了。说明：在此之前也试了其他一些内核版本如：2.6.7，2.6.11。但是感觉补丁做的不太完善，需要自己做的还是太多，所以最后选择了 2.6.9。据说 2.6.14 不错，用空了去试试。

第二节打补丁，编译。

环境弄好了，当然迫不及待了，先把补丁打上。把 linux-2.6.9 拷贝到 uClinux-dist 目录下。改个名字 linux-2.6.x.没有什么道理。都这样做的。现在你的两个 patch 在什么位置啊。我的是在/home/xcj/ ,然后你来到 linux-2.6.x 的目录 patch -p1 < ../../linux-2.6.9-uc0.patch。打一个，再打一个：patch -p1 < ../../linux-2.6.9-uc0-hsc0.patch，好了补丁弄好了。编译吧。Make menuconfig.显示没有 2.6.x。哦，原来是 verdor 目录下没有 config.linux-2.6.x 文件。在 arch/armnommu/config 里面已经有了 s3c44b0x_defconfig,copy 过去改个名字 config.linux-2.6.x..好了。配置一下。



在 Linux Kernel Configuratin 中的 System Type 菜单中一定要按照开发板的硬件进行正确配置，主要是 flash size ,base sdram size,base 等，Arm Core Clock 为 6000000 等。

好了，这下编译，2.6.X 不需要 2.4.x 的那么多步骤，直接 make.完了，去 image 目录看看，哎呀，没有 uclinux_ram.bin 啊？什么原因呢，哦修改 top makefile 就是 uclinux-dist 目录下的哪个 makefile.在 21 行的目标 all:那个地方改为 all: subdirs romfs modules modules_install image linux images，然后象 romfs 那样在 266 行加上.PHONY:linux 等。因为 2.6.x 和 2.4.x 不一样，你只要敲一个 make 所以你要把那些目标再定义一下。不象 2.4.x 里你要 make lib_only make user_only make romfs 等。还有，在 verdor/samsong 目录下的 makefile 也要改，要不 makefile 会报错，加上 images 的定义就行了。好了 makefile 搞定了。这下 make；怀这忐忑的心情，哈，有了，在 image 目录下已经有 uclinux_ram.bin.gz,运行一下。靠，除了 Starting application at 0x0c008000 ...什么都没有了这应该是波特率的问题了。查了一下，原来 flash,sdram 的大小没有配对，修改一下。Make 好了，运行这下有了………

第三节 romfs 的问题。

运行后出现：Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block，无法挂载根文件系统。由于 2.6 没有了 blkmem 所以根文件系统的做法和 2.4 的有所差别。经过查阅资料：
修改 linux-2.6.x/arch/armnommu/arch/kernel/vmlinux.lds.S，添加 romfs.o

81 *(.got) /* Global offset table */
82 romfs_start = .;
83 romfs.o
84 romfs_end = .;

修改 linux-2.6.x/arch/armnommu/kernel/setup.c，添加变量 romfs_start,romfs_end 及设置 default_command_line

64 extern int _stext, _text, _etext, _edata, _end;
65 extern int romfs_start,romfs_end;
683 char *from = default_command_line;
685 sprintf(default_command_line, "root=/dev/ram0 initrd=0x%08lx,%ldk keepinitrd", (unsigned long)&romfs_start,((unsigned long)&romfs_end - (unsigned long)&romfs_start)>>10);

由于 root 文件系统使用/dev/ram0，因此 rc 时就不能使用/dev/ram0，而使用/dev/ram1, /dev/ram2，因此同时修改 vendors/Micetek/44b0/rc 文件. 2 /bin/expand /etc/ramfs.img /dev/ram1

好了，编译运行。还是 Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block，什么问题，经过查看，原来 2.4.x 的时候/bin 下的应用程序太多了，超过了我们的 ram0 的大小，在 core application 中减掉一些，编译，运行，好了，可是又出现了：Warning: unable to open an initial console. Kernel panic - not syncing: Attempted to kill init。

第四节：串口，console 的问题。

Unable to open an initial console .这个问题我是这样解决的，其中有我自己的理解，有什么问题可以给我改正。经过查阅资料，明白了。

原因：在安装 REDHAT 时选择了完全安装后，即选择 Custom，然后在 Package 中选择 everything 会出现此种现象，这是因为光盘提供的开发环境使用的是/usr/local/bin/genromfs，但 PC 机器上也自带了一个 genromfs，这两个是不同的。但 PC 机的搜索路径中/usr/bin/优先于/usr/loca/bin/，因此编译是使用了宿主 PC 机的 genromfs，而不是 uClinux 所要使用的，所以会出错。

解决办法有两种：

(1) 在 PC 机上执行：

cp -f /usr/local/bin/genromfs /usr/bin

（2）修改 uClinux/vendor/ROMFS/Makefile

将其中的 genromfs 改为/usr/local/bin/genromfs 即可。

我选择了第 2 种方法，解决了，那 Attempted to kill init。呢，那是 inittab 上什么都没有造成的，加一个 console:linux:/bin/sh 好了，两个问题都解决了，但是 console 是用于内核空间访问的，shell 启动后要输入输出就要自己做串口驱动，经过千辛万苦，根据别人已经有的改改，（见源程序）这里需要说明的是 2.6.9 本身给我们提供了一个 console 驱动，但是哪个只是内核级的很简易的串口，你的 shell 是访问不到的（想想：为什么？）因为：接口，linux 的驱动接口。这个没有道理的。好了这下可以在串口上运行 shell 了

故事才刚刚开始。。。。。。。。。

第五节：网卡

网卡，我们的开发板用的是 ne2000 的兼容网卡。2.6 的内核中，驱动部分的接口改变的很多，还好，这部分还不算复杂。网上都有很多介绍。主要就是改 baseaddr 和 irq 这两件事情。主要的文件包括：ne.c 8390.c 8390.h space..c net_init.c. 由于 ne2000 的兼容网卡比较丰富所以驱动做的很复杂，包括 pci，等总线的支持。总的来说 8390.c 是提供上层调用的接口，具体的网卡收发是在 ne.c 中做的。在 ne.c 的 do_ne_probe(dev)中加上：

```
    #elif defined(CONFIG_BOARD_MBA44)
    static int once = 0;
    if (once)
        return -ENXIO;
    dev->base_addr = base_addr = 0x06000000;//NE2000_ADDR;
    dev->irq = 24;//NE2000_IRQ_VECTOR;
    once++;
    printk(KERN_ERR "ne.c do_ne_probe(): Probe at %#lx .\n", dev->base_addr);
#endif
```

当然你还要在合适的地方定义 CONFIG_BOARD_MBA44，很简单就在 kconfig 中吧，参照写一句：CONFIG BOARD_MBA44。。。。。。。。就 ok 的。现在就开始设置你自己的网卡工作模式和偏移地址。在 8390.h 中。加上：

```
#define CONFIG_NE2000_16b
#ifdef    CONFIG_NE2000_16b
#define    ETH_ADDR_SFT  (1)
#else
#define    ETH_ADDR_SFT  (0)
#endif          //add by xcj
#elif defined(CONFIG_ARM_ETHERH) || defined(CONFIG_ARM_ETHERH_MODULE)
#define EI_SHIFT(x)     (ei_local->reg_offset[x])
#undef inb
#undef inb_p
#undef outb
#undef outb_p

#define inb(_p)         readb(_p)
```

```
#define outb(_v,_p) writeb(_v,_p)
#define inb_p(_p)   inb(_p)
#define outb_p(_v,_p)    outb(_v,_p)

#elif defined(CONFIG_NET_CBUS) || defined(CONFIG_NE_H8300) ||
defined(CONFIG_NE_H8300_MODULE)
#define EI_SHIFT(x)    (ei_local->reg_offset[x])
#else
#define EI_SHIFT(x)    ((x)<<ETH_ADDR_SFT)
#endif
```

这个的意思就是定义你自己的地址访问方式了。同样在 ne.c 中：

```
#define NE_DATAPORT      (0x10<<ETH_ADDR_SFT)   /* NatSemi-defined port window offset. */
#define NE_RESET      (0x1f<<ETH_ADDR_SFT)   /* Issue a read to reset, a write to clear. */
#define NE_IO_EXTENT      (0x20<<ETH_ADDR_SFT) //add by xcj.
```

在 space 的 static struct devprobe2 isa_probes[] __initdata：中已经定义了 probe 函数，所以基本上问题就解决了。编译一下，运行：好的，网卡找到了，中断也没有问题！太棒了！ping 一下，靠,根本 ping 不通。找原因啊，找，找，找，原来是：我 ping 的时候在 linux 这边用:tcpdump –w arp 监视到的数据是：FF FF FF FF FF FF FF FF FF FF FF FF 00 00 49 49 34 34 00 00 08 08 06 06，哦是多 发了一个字节的数据，弄啊弄，找到这样一句话：

熟悉 ARM 家族的人应该知道 ARMv3 和 ARMv4 的一些区别，看看这两汇编，就可以开出来他们对 16 位数读写操作的不同，按照道理 S3C44B0 应该是 ARMv4（我记得应该是，不到出处了，至少看了那个两汇编文件，我认定应该用 ARMv4 那个），可是，看了一下便一输出的.o 文件，是 io-writesw-armv3.o，显然弄错了，这里就是问题了。那么为什么要编译 ARMv3 而不是 ARMv4 这个文件呢？在 Makefile 和 Config.in 中经过一番寻找，终于找，原来在定义 arch/armnommu/mm/kconfig 中，

```
config CPU_ARM710
    bool "Support ARM710 processor"
    depends on ARCH_S3C3410 || ARCH_ATMEL || ARCH_S3C44B0
    default y if ARCH_S3C3410 || ARCH_ATMEL || ARCH_S3C44B0
    select CPU_32v3
    select CPU_CACHE_V3
    help
```

那么，默认情况下，就定义 CONFIG_CPU_32v3，用它来编译。好了.改成 ARMv4 编译运行，ok,ping ,tftp 没有问题了，但是有个问题：有时会出现：eho time out.，后来分析：驱动和 2.4 的一模一样，编译器也是一样的。什么原因呢：估计是 44b0 和网卡时序的配合问题。最后没有办法了，2.6.9 自带了一个网卡驱动，改了一下，irq 和 addrbase

呵，呵，好了。没有这个问题了。就先用这个吧。据说 2.6.14 的网络比较好用，下一步用 2.6.14 看看。好了网卡算是好了，加上 nfs 吧，在 fs 的配置中加上：

CONFIG_NFS_FS y
CONFIG_NFS_V3 y

你必须在 filesystem 的 nfs filesyste support 中打开 nfsv3 client support,然后重新编译内核;

另外在应用程序中添加 Network->Applications->portmap 以及 Filesystem Application->mount/umount 或者 BusyBox->mount/umount.

第六节 如何在 JX44B0 中添加 nfs

内核支持:

1,网络设备正常;

2,TCP/IP 协议能够工作;

3,File System->Network file systems -> NFS File System support

4,File System->Network file systems ->NFSV3 client support

应用程序支持:

1,添加 Network->Applications->portmap

2,添加 Filesystem Application->mount/umount 或者 BusyBox->mount/umount,两者必须使用一个.

重新编译内核,然后将内核下载到目标系统:

输入如下命令:

#portmap&

#mount –t nfs 192.168.1.180:/tftpboot /mnt

#ls /mnt

15,nfs mount 总是出现错误 mount: RPC: Timed out

答:

1,请设置服务器防火墙为"无防火墙";

2,如果使用直连网线和主机连接,请将目标机和主机得 ip 设置到同一个网段,如 192.168.1.xx,且将主机网关去掉.。

好了进入/var,mkdir mnt munt –t nfs:192.168.0.4:/tftpboot/ /var/mnt/

Nfs 成功了。

第七节：lcd

移植 lcd 出现了很多问题。主要是因为 2.6 的接口和 2.4 的相比变的太多了。2.6 的内核把 console 和 fb 分开了。2.4 中 fbmem.c 主要提供给文件系统接口，使 user 可以通过统一的文件层次上访问 framebuffer. 包括 fileoperaton 的实现，其中当然又做了很多的二次跳转，这些跳转就是你的 lcd 驱动程序要实现的了。Fbcon.c 的函数主要是提供给内核访问。同 console.c vt.c 接口。在 2.4 中 fbmem.c 和 fbcon.c 的耦合是通过 display 数据结构联系的，在 2.6 中，display 被削弱了，所以很多关于设备自身的东西都在 fbmem.c 和你自己的驱动中.skeletonfb.c，tgafb.c 提供了一些范例可供参考：可以看出我们的驱动要提供给或者说要向 fbmem.c 注册自己的方法：

```
static struct fb_ops tgafb_ops = {
	.owner			= THIS_MODULE,
	.fb_check_var		= tgafb_check_var,
	.fb_set_par		= tgafb_set_par,
	.fb_setcolreg		= tgafb_setcolreg,
	.fb_blank		= tgafb_blank,
	.fb_fillrect		= tgafb_fillrect,	needed
	.fb_copyarea		= tgafb_copyarea,	needed
	.fb_imageblit		= tgafb_imageblit,	needed
	.fb_cursor		= soft_cursor,		needed
};
```

其中.fb_fillrect .fb_copyarea .fb_imageblit .fb_cursor 是必须的。其意义可以从文字上看出。其中:fb_cursor 是和鼠标相关的，fb_imageblit 是把控制台过来的字符等画在你的 lcd 上，这些函数又分别在 cfbcopyarea.c cfbfjillrect.c cfbimgblt.c 中，而我们的 lcd.c 文件只是提供上层 open ,write 等时的数据内存分配，和通用的 lcd 参数设置，比如用户可以在用户空间设置调色板等。现在我们的重点就是要正确

配置我们的 lcd 后进入到 fb_imageblit 和 fb_cursor 中去根据我们自己的 bpp 来正确显示文字和光标。在 cfbimgblt.c 中，可以看到 2.6 的核支持了 cfb_tab[8],cfb_tab[16]的快速显示，但是我们的屏是单色，4bpp, 所以就参照 2.4 的内核加了自己的显示。而光标的显示：2.6 和 2.4 的也差不多，都是利用了内核的定时器来触发，不停的画，不停的删，所以就给用户了闪烁的感觉。其坐标是根据 vt.c 里的输出不断调整的。需要注意的是光标最后的绘制也是调用 cfb_imagblit,所以还是根据 2.4 的显示来绘制了光标。因为如果按照 2.6 默认的处理在 4bpp 下，提取字模的时候信息会丢失，显示出来的字就是空的，很模糊。很毛造。
下面给出几个 lcd 的参考资料：

附如：A

S3C44B0X 的 LCD 控制器简述

东北大学　胡楠

控制器简介

在复杂的 PC 机中，我们经常提到显示卡这个东西，相信大家对显示卡的原理都不陌生。LCD 控制器就相当于嵌入式系统的显示卡。它负责把显存中的 LCD 图形数据传输到 LCD 驱动器，并产生必须的 LCD 控制信号。显存与系统存储器共用主存空间。这样做有几个好处：节约存储器，提高空间利用率，符合嵌入式系统的设计精神；显存设置在主存空间内，对显存的操作，实际上就是对主存的操作，简化程序编制。

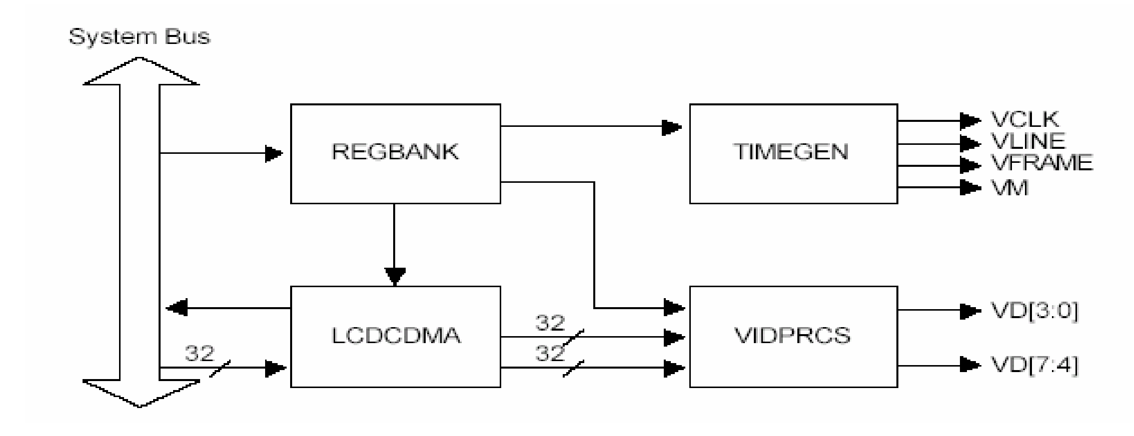


图 1 控制器系统框图

LCD 控制器使用 LCDCDMA，把显存数据输入 LCD 控制器。LCDCDMA 是专用 DMA，在不用 CPU 参与的情况下，自动传输显存的视频数据到 LCD 控制器。控制器中有一个 24 字的 FIFO 存储区。其中 12 个 FIFOL，12 个 FIFOH，在单扫描模式下仅 12 个 FIFOH 可用。当 FIFO 为空或部分为空时，LCDCDMA 请求从显存预取数据（使用突发传输模式，一次预取 4 个字；在传输期间，不允许出让总线控制权）。

S3C44B0 的 LCD 控制器使用时间抖动算法和帧速控制方法，实现 LCD 上的单色、4 级灰度（每个象素占用 2 位）、16 级灰度（每个象素占用 4 位）显示，也能与彩色 STN 接口，支持最大 256 色（每个象素占用 8 位）的显示 。

LCD 控制器可以编程支持不同水平和垂直点数（640×480， 320×240， 160×160 等等），以及不同的接口时序和刷新速率的 LCD，支持 4 位双扫描、4 位单扫描、8 位单扫描的 LCD 显示器，并支持虚拟屏幕，实现硬件水平/垂直卷动。（这些概念后边会详细解释）

灰度显示中，4 级灰度显示模式使用查找表，允许在 16 级灰度中选择 4 级灰度显示。该查找表和彩色查找表的蓝色查找表公用一个寄存器 BULEVAL[15:0]，灰度 0 由 BLUEVAL[3:0]值表示，灰度 1 由 BLUEVAL[7:4]值表示，灰度 2 由 BLUEVAL[11:8]值表示，灰度 3 由 BLUEVAL[15:12]值表示。16 级灰度显示模式不使用查找表。

彩色 8 位显示模式中，3 位分配为红，3 位绿，可以同时显示 8 个红色与 8 个绿色，2 位蓝色位，可以

同时显示 4 个蓝色，合起来最大显示 256 色。红、绿、蓝分别使用不同的查找表，红色、绿色查找表入口都是 32 位（并分成 8 组），分别由 REDVAL[31:0]、GREENVAL[31:0]寄存器指示，蓝色查找表入口为 16 位，由 BLUEVAL[15:0] 寄存器指示，并分成 4 组。也就是说红色、绿色可以在 16 种颜色组合中选择 8 色进行显示，蓝色可以在 8 种颜色组合中选择 4 色进行显示。

LCD 自刷新模式：S3C44B0X 支持 LCD 自刷新模式，以减少电源消耗，这时电源管理可以进入 SL_IDLE 模式。

控制器原理

扫描

LCD 控制器的一项重要工作是向 LCD 驱动器，按照一定的格式输出显示数据，然后驱动器逐象素的显示，由于人眼的滞留效应，看到的是连贯的图像。这个过程称为扫描。控制器的 VD[7..0]是数据输出端口。根据不同的驱动器，分为 4 位双扫描、4 位单扫描和 8 位单扫描，位数指输出端口的位宽，单扫描同时刷新屏幕上的一行，双扫描同时刷新两行。

VD3 VD2 VD1 VD0 VD3 VD2 VD1 VD0 . . . . . .

VD7 VD6 VD5 VD4 VD7 VD6 VD5 VD4 . . . . . .

4-bit Dual Scan Display

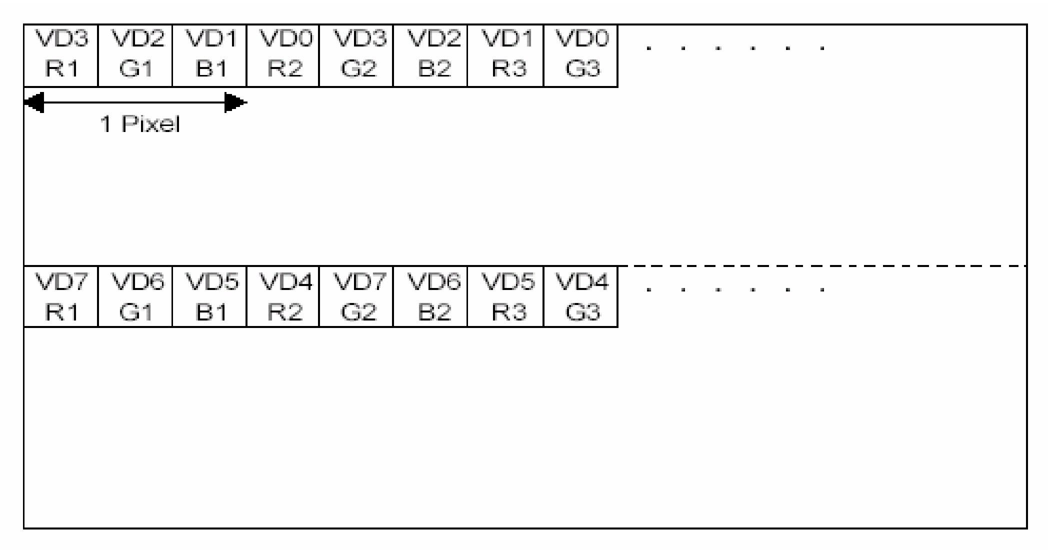
VD3 VD2 VD1 VD0 VD3 VD2 VD1 VD0 . . . . . .

4-bit Single Scan Display

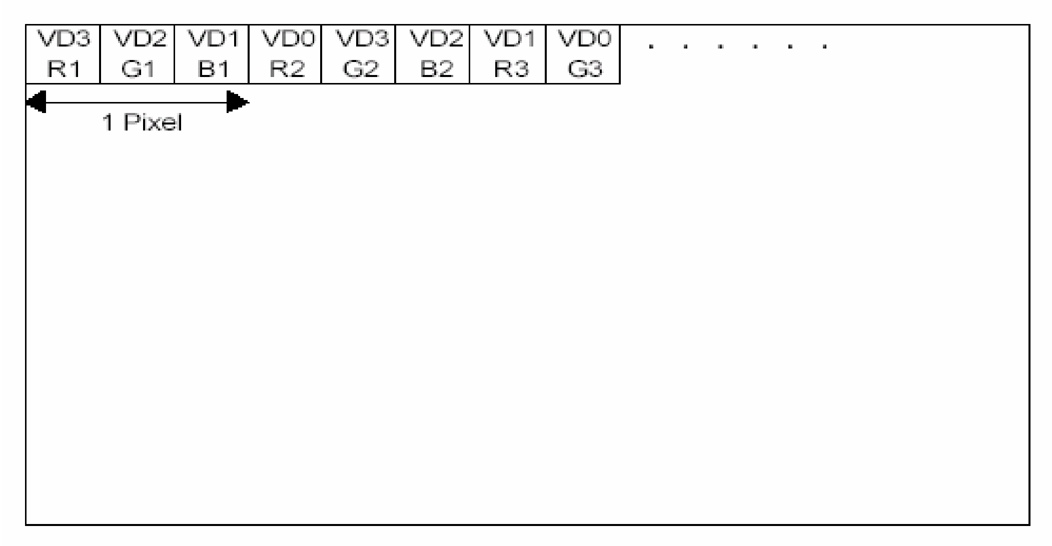
VD7 VD6 VD5 VD4 VD3 VD2 VD1 VD0 . . . . . .

8-bit Single Scan Display

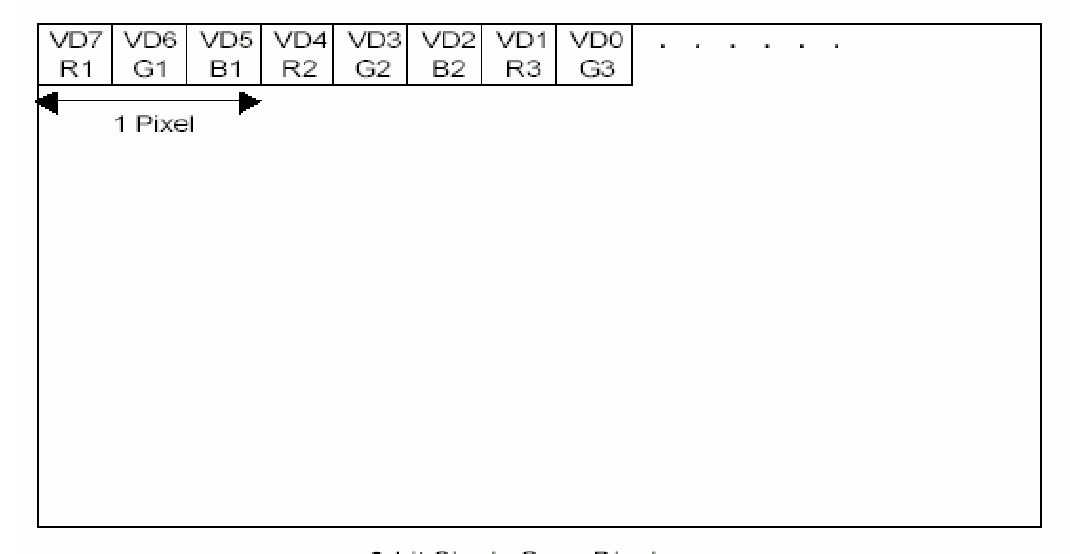
图 2-1 扫描模式



4-bit Dual Scan Display



4-bit Single Scan Display



8-bit Single Scan Display

图 2-2 彩色扫描模式

当然，驱动器光有数据还不能显示，必须知道什么时候这些数据有效，什么时候换行，什么时候换页，等等。这就需要一些同步信号。控制器提供 VCLK/VLINE/VFRAME/VM 信号做同步用。在 VCLK 上升延，VD 输出数据；VCLK 下降延，驱动器锁存该数据；所以 VCLK 实际是象素的同步信号。同理，VLINE 是行的同步信号，VFRAME 是帧的同步信号。VM 是驱动器切换板电极的交流信号。切换的速率是可以编程的，由 MVAL 寄存器控制。

VM Rate = VLINE Rate / (2 ×MVAL)
VCLK(Hz)=MCLK/(CLKVAL× 2)

此外需要设置 LCD 的分辨率和显示模式。因为这些参数和控制器的同步信号直接相关。举例说明：一个 LCD 面板是 320×240 的分辨率，那么控制器每输出 320 个象素，就会输出一个行同步信号；每输出 240 行之后，就应该输出帧同步信号；如果是 4bit 单行扫描，那么每个 VCLK 可以输出 4 个象素，则每行只会有 320÷4＝80 个 VCLK 信号。可见前面的结论换句话说，LCD 的分辨率和显示模式的参数，决定了控制器的时钟。

HOZVAL = (Horizontal display size / Number of the valid VD data line) -1
彩色：Horizontal display size = 3 * Number of Horizontal Pixel

LINEVAL = (Vertical display size) -1：单扫描
LINEVAL = (Vertical display size / 2) -1：双扫描

查找表

既然控制器支持 16 级灰度，那么还要 4 级灰度干吗呢？何不舍弃低级的 4 级灰度，都使用更高级的 16 级灰度呢？事实并非如此。16 级灰度需要 4bit 来表示一个象素，而 4 级灰度只需要 2bit 就够了。所需的位数越少，就越节省显存空间（使用更少的存储器），操作显存耗费的 CPU 资源也越少，这样系统功耗必然随之减小。因为很多应用中对灰度级要求不高，最好根据需求，衡量使用什么样的显示方式。

有些比较特殊的应用，虽然只使用 4 级灰度，但是这 4 级灰度并不是线性的。线性这个概念在这里怎么理解呢？打个比方，就好像拿着黑色颜料和白色颜料混合。如果是线性的 16 级灰度，第 15 级就是只用黑色颜料；第 0 级则只有白色；第 8 级相当于 8 分黑色、7 分白色均匀混合；依此类推。使用 4 级非线性灰度，就是从 16 级灰度中可编程的挑选出来 4 种。例如在某个应用中，要求象素点能够完全熄灭、能够达到最大亮度，以及正常显示文字；我们可以设置 4 级非线性灰度，第 0 级相当 16 级线性灰度的第 0 级（熄灭），第 3 级相当 16 级的第 15 级（最亮），第 1 级和第 2 级都相当 16 级的第 11 级（普通文字）。

查找表就是为了适应非线性灰度而设置的一种数据结构，它其实是一组寄存器。拿前边的例子来说，在 4 级非线性灰度模式下，LCD 控制器读到一个象素是第 1 级灰度，它就会到查找表中，查找第 1 级在 16 级线性灰度中的对应级别，结果是 16 级的第 11 级。完成这种转换之后，控制器按第 11 级输出。

顺便说一下，在彩色模式下也有查找表。44B0 用 8 位表示彩色，3 位红色，也就是 2^3＝8 级红色，我们偷换概念，姑且叫“红度”吧。但是控制器可以输出 16 级“红度”，16 级“绿度”，8 级“蓝度”，所以再次用到查找表。

虚拟屏幕

我们在 PC 上编写文档的时候，一屏幕显示不下，通过滚动翻页来浏览全部文档。虚拟屏幕就好像能容纳整个文档的大屏幕，但显示器只对应其中的某一部分。结合下面的图形很容易理解。

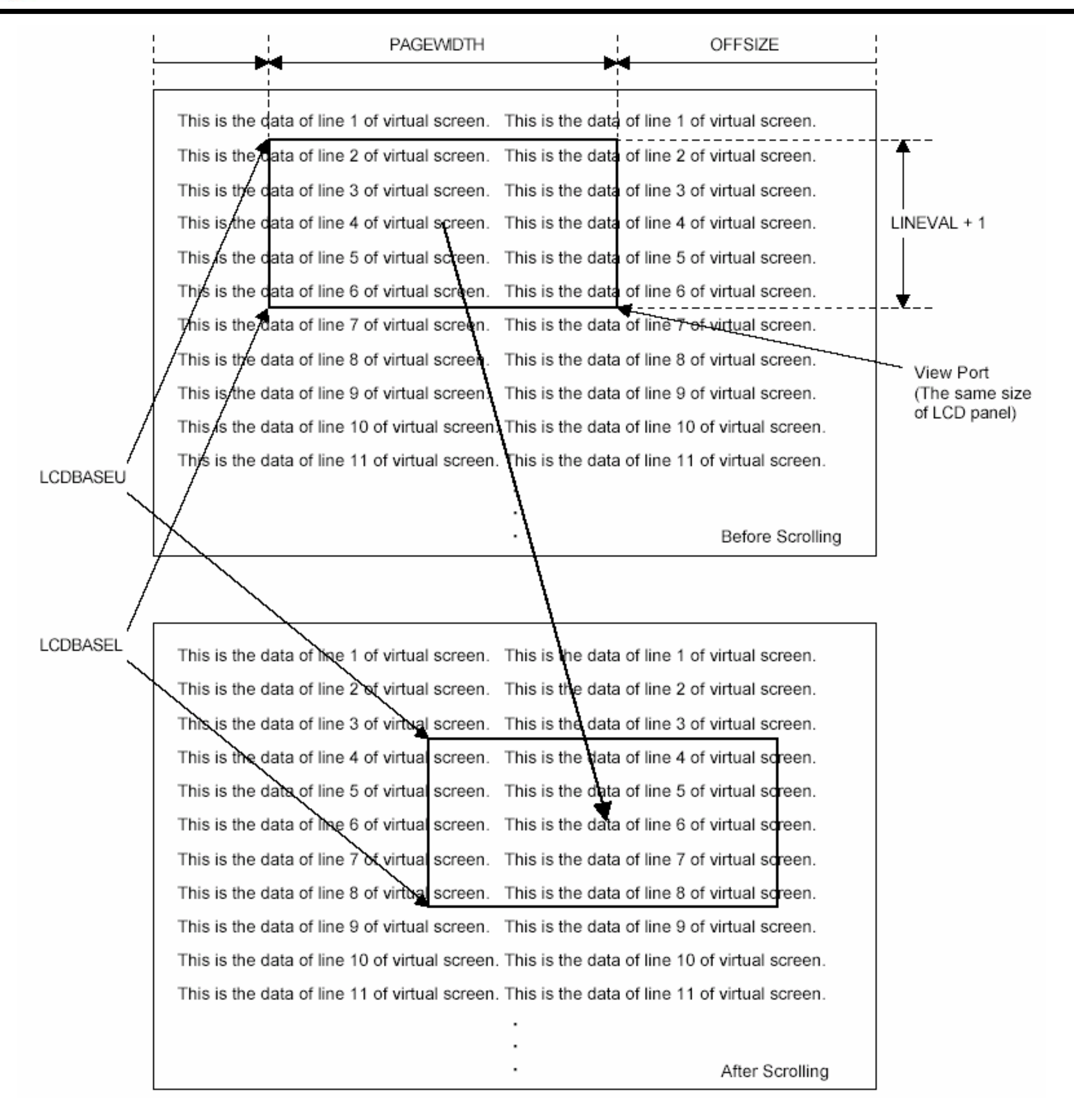


图 3 虚拟屏幕

虚拟屏幕从原理上讲是一个内存的概念。因为程序员对显示器的操作实际上是对显存的操作，把代表显示器每个象素的数据写入显存的过程。对于虚拟屏幕，程序员维护着一个更大的屏幕，更多的显存而已。事实上，虚拟屏幕的大小的确受显存大小的限制。实际显露出来的显示器部分，其起始点（左上角）由程序控制，改变这个点的位置，显示器就会在虚拟屏幕上移动。

在图 3 里，有几个参数需要说明。PAGEWIDTH 是真实显示器的宽度；OFFSIZE 是真实屏幕左端到虚拟屏幕最右端的距离；LCDBASEU 是真实显示器起点对应的显存地址；LCDBASEL 只用在双扫描模式，是下半帧对应显存的开始地址，当然在单扫描时就没有什么用处了。

LCDBASEL =LCDBASEU + (PAGEWIDTH + OFFSIZE)×(LINEVAL +1)

程序通过改变 LCDBASEU 和 LCDBASEL 的值来滚动屏幕。但在一帧刷新结束时，不能改变 LCDBASEU 和 LCDBASEL 的值。因为一帧结束时，LCDCDMA 已经预取了下一帧的数据。如果这时改变显存中的帧的内容，之前预取的数据将无效，从而显示也不正确。控制器提供一个 LINECNT 寄存器，他会从 LINEVAL 向 0 倒计数，记录帧已经刷新了多少行。通过检查 LINECNT，可以避免在不恰当的时间更改显存。检查的时候，中断应当被屏蔽，否则在读出 LINECNT 后，如果某个中断刚好执行，LINECNT 的值可能会改变。

抖动算法

S3C44B0X 利用抖动算法实现灰度级和 STN 的彩色显示。

时常会有疑问：16 级灰度，需要 4bit 来表示一个象素，为什么控制器只使用了 1bit 输出就实现了？256 色需要 8bit 表示一个象素，为什么只用 3bit 输出就可以呢？下面讲讲抖动算法，大家就会明白这些道理。

LCD 屏幕以一个很快的（通常为 80Hz）速度刷新，因为众所周知的人眼滞留效应，感觉上是流畅的画面。一个帧周期内，象素只有点亮和熄灭两种状态（44B0 使用的简单的 LCD 驱动器支持两种状态，但是 TFT 等可以支持更多的状态）。正是基于这种原理，当象素一亮一灭，且切换的频率很低时，看到的是一闪一闪的点；如果切换频率很高，人眼觉察不出来他在闪烁，只会感觉没有那么亮罢了。屏幕不断快速刷新，亮的次数多，熄灭的次数少，看起来就比较重；反之看起来比较淡。这种颜色的浓淡变化构成不同灰度级别。可以用“占空比”来量化表示，就是亮的次数占刷新次数的比例。表 1 显示出不同灰度级对应的占空比。

表 1 不同灰度级对应的占空比

Pre-dithered Data (Gray Level Number)	Duty Cycle	Pre-dithered Data (Gray Level Number)	Duty Cycle
15	1	7	1/2
14	6/7	6	3/7
13	4/5	5	2/5
12	3/4	4	1/3
11	5/7	3	1/4
10	2/3	2	1/5
9	3/5	1	1/7
8	4/7	0	0

Pixel Duty Rate

举例说明，假设某个象素是第 11 级灰度，表 1 中查出占空比为 5/7。控制器便自动以每 7 帧为一个周期，其中 5 帧该象素亮，另外 2 帧熄灭。这就解释了为什么每个象素只用 1bit 输出实现 16 级灰度。

前面把彩色分成“红度”、“绿度”、“蓝度”，使用和灰度输出相同的思路，用 3bit 可以实现彩色输出。

这时又出现一个新问题。极端的例子，比如 1 秒内，0.5 秒点亮，0.5 秒熄灭，虽然占空比依然是 1/2，但是很显然看起来不会是第 7 级灰度（第 7 级灰度对应占空比为 1/2），而是闪烁的画面。所以，应该把亮灭状态离散化，尽量不要让同一个象素在相邻帧保持相同状态。

与此同时，这种现象也影响同一帧内相邻的象素。图像的灰度级就像地图上的等高线，相邻象素的灰度很大几率是相同的。如果四条输出线总是输出相同的数据，显示就会出现局部闪烁的现象。解决方法依然是离散化。

考虑到上述两个问题，就可以着手计算输出序列。当然这是一个数学问题，要根据不同的应用不断的优化。一般可以采用经典的序列。好在 44B0 的 LCD 控制器并没有把抖动算法的输出序列固化在芯片里，可以通过寄存器更改。